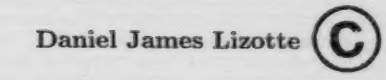
University of Alberta

Practical Bayesian Optimization

by

Daniel James Lizotte 

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Computing Science

Edmonton, Alberta
Fall 2008

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-46365-9
Our file Notre référence
ISBN: 978-0-494-46365-9

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

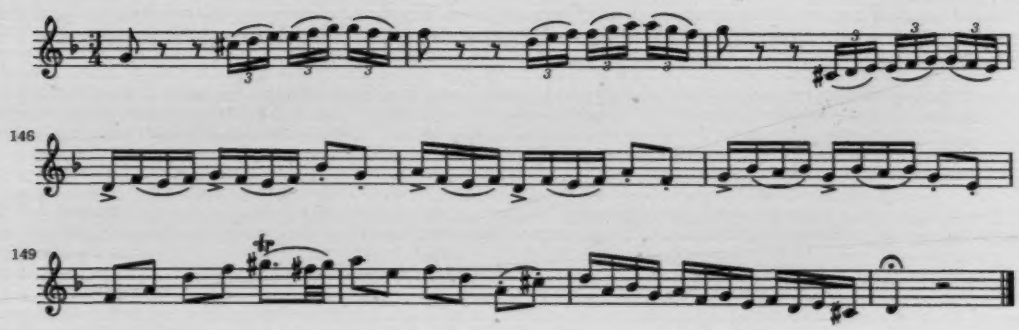
L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

–J. S. Bach

Abstract

Global optimization of non-convex functions over real vector spaces is a problem of widespread theoretical and practical interest. In the past fifty years, research in global optimization has produced many important approaches including Lipschitz optimization, simulated annealing, homotopy methods, genetic algorithms, and Bayesian response-surface methods. This work examines the last of these approaches. The Bayesian response-surface approach to global optimization maintains a posterior model of the function being optimized by combining a prior over functions with accumulating function evaluations. The model is then used to compute which point the method should acquire next in its search for the optimum of the function. Bayesian methods can be some of the most efficient approaches to optimization in terms of the number of function evaluations required, but they have significant drawbacks: Current approaches are needlessly data-inefficient, approximations to the Bayes-optimal acquisition criterion are poorly studied, and current approaches do not take advantage of the small-scale properties of differentiable functions near local optima. This work addresses each of these problems to make Bayesian methods more widely applicable.

Acknowledgements

Seeing a doctoral degree through from beginning to end is a monumental individual and collaborative effort. Here, I would like to explicitly express my thanks to some of those who helped me along my way.

Thank you Russ Greiner for exposing me to so many different ideas and problems, and for making sure I know what I'm talking about.

Thank you Dale Schuurmans for your faith in me, and for encouraging me to pursue my own interests, and for teaching me never to give up.

Thank you Mike Bowling for being a colleague and a friend, and for your perspective, and for making me a "robot scientist."

Thank you Tao Wang for the immense amount of work you put in during our research collaborations.

Thank you Colin Cherry for being my best friend, and for realizing the practical potential of this research.

Thank you Daniel Neilson for your crucial contribution to ensuring this thesis has an impact on the computer vision community.

Than you Ruth Shaw and Larry Garey for teaching me to love research in my first years of university education.

Thank you to all of my friends, who are are happily too many to list here.

Thank you to my family for your unwavering support.

Contents

1 Introduction 1
- 1.1 Objective and Contributions . . . 2

2 Background 3
- 2.1 Optimization in $\mathbb{R}^d$ . . . 3
  - 2.1.1 Local Optimization . . . 4
  - 2.1.2 Non-Bayesian Methods . . . 5
  - 2.1.3 Introduction to the Bayesian Approach to Optimization . . . 8
  - 2.1.4 Preview of Gaussian Processes . . . 9
  - 2.1.5 A History of Bayesian Optimization Methods . . . 11
- 2.2 Gaussian Processes in Machine Learning . . . 23
- 2.3 Extrema of Gaussian Processes . . . 25
  - 2.3.1 Spectral representation . . . 27
  - 2.3.2 Stationary processes in one dimension . . . 28
  - 2.3.3 Stationary processes in many dimensions . . . 31
  - 2.3.4 The Euler characteristic of excursion sets . . . 33

3 Practical Bayesian Optimization 39
- 3.1 Gaussian Process Model . . . 39
  - 3.1.1 Parameter Learning . . . 39
- 3.2 Acquisition Criteria . . . 49
- 3.3 Summary . . . 56

4 An Experimental Framework for Global Optimization 63
- 4.1 Models . . . 64
  - 4.1.1 Kernels . . . 64
  - 4.1.2 EEC, Complexity, and Dimensionality . . . 65
- 4.2 Using Derivatives: Connections with Local Search . . . 66
  - 4.2.1 Incorporating Derivative Information . . . 68

5 Empirical Results 70
- 5.1 Experimental Setup . . . 70
  - 5.1.1 Test Model Choices . . . 71
- 5.2 Discussion . . . 71
  - 5.2.1 The Effect of ξ_r . . . 72
  - 5.2.2 The Effect of Priors on Performance . . . 76
  - 5.2.3 Effect of Mismatched Models . . . 78
  - 5.2.4 The Effect of Gradient Information . . . 79
- 5.3 Summary . . . 81
- 5.4 Functions from the Literature . . . 81

6 Applications **89**
6.1 AIBO Gait Optimization 89
6.2 Stereo Matching Parameter Optimization 97

7 Conclusion **102**
7.1 Contributions 102
7.2 Future Work 103
7.2.1 Reducing Computational Requirements 103
7.2.2 Application to Translation Directed Word Alignment 104
7.3 Summary 105

A Complete Experimental Data **106**
A.1 Results using 2D squared exponential test kernel, equal length scales 106
A.2 Results using 2D Matérn test kernel, equal length scales 111
A.3 Results using 2D squared exponential test kernel, unequal length scales 116
A.4 Results using 2D Matérn test kernel, unequal length scales 121
A.5 Results using 8D squared exponential test kernel, unequal length scales 126
A.6 Results using 32D squared exponential test kernel, unequal length scales 131
A.7 Results using 2D squared exponential test kernel, equal length scales, Matérn optimization model kernel 136
A.8 Results using 2D Matérn test kernel, equal length scales, squared exponential optimization model kernel 141
A.9 Results using 2D squared exponential test kernel, equal length scales, gradients included 146
A.10 Results using 2D Matérn test kernel, equal length scales, gradients included 151

B Effect of Priors on Performance **156**
B.1 With function values only 156
B.2 With gradient observations 163

List of Symbols

f	The function to be optimized. $f : \mathcal{X} \subseteq \mathbb{R}^d \to \mathbb{R}$
$\mathcal{X}$	The domain of f
x, z, w	Points in the domain of f
x_j	The jth coordinate of the point x
$\|\|x\|\|$	The Euclidean norm of x
$\mathbf{x}$	A list of points $x^1, x^2, ..., x^n$
$x^{1:k}$	A sublist of points $x^1, x^2, ..., x^k$
$f(\mathbf{x})$, or $\mathbf{f}$	A list of function values $f(x^1), f(x^2), ..., f(x^n)$
$k(x, z)$	The kernel function between the points x and z
$k(\mathbf{x}, z)$	A column vector whose ith element is $k(x^i, z)$
$k(x, \mathbf{z})$	A row vector whose ith element is $k(x, z^i)$
$k(\mathbf{x}, \mathbf{z}) = \mathrm{K}$	A kernel matrix whose (i, j)th element is $k(x^i, z^j)$
$\mu(x)$	The prior mean function
σ_f^2	The "signal" or "process" variance of a Gaussian process
σ_n^2	The noise variance of a Gaussian process
ℓ_i	The characteristic length-scale along the ith dimension
L	A diagonal length-scale matrix
X	A random variable
$p(X = x)$	Probability (or density) of the event $X = x$
$\mathrm{E}[X]$	The expectation of X
$\mathrm{Cov}(X, Y)$	The covariance between X and Y: $\mathrm{Cov}(X, Y) = E[(X - E(X)) \cdot (Y - E[Y])]$
$(a)_+$, $[a]_+$	$\max(a, 0)$
I	The identity matrix
$\mathbb{R}$	The set of real numbers
$\Psi(x)$	The standard normal right tail probability $p(X > x)$, $X \sim \mathcal{N}(0, 1)$
$H_k(x)$	The kth "probabilist's" Hermite polynomial Some properties of $H_k(x)$: $H_k(x) = (-1)^k \cdot \mathrm{e}^{x^2/2} \cdot \frac{\mathrm{d}^k}{\mathrm{d}x^k} \mathrm{e}^{-x^2/2}$ $H_0(x) = 1$ $H_1(x) = x$ $H_{k+1}(x) = xH_k(x) - kH_{k-1}(x)$

Chapter 1

Introduction

We are interested in a staggeringly difficult yet beautifully concise problem:

$$\max_{x \in \mathcal{X}} f(x) \tag{1.1}$$

For example, the function f could be the speed of a walking robotic dog, or the quality of a depth map reconstructed from a stereo image pair. The set $\mathcal{X}$ could be the space of possible motor commands the robot understands, or the space of parameters that define how similar two pixels are. They could also represent the objectives and domains needed to solve most of the interesting problems in computing science, from linear regression to maximum satisfiability.

In order to solve (1.1), we will need to know something about f. To encode this information, we will appeal to the axioms of probability theory.

$$P(f|\mathcal{F}) \propto P(\mathcal{F}|f)P(f) \tag{1.2}$$

Specifically, we will make use of (1.2), widely known as *Bayes's Rule*. This will allow us to reason about the specific f we are dealing with by combining two quantities: Our "prior" belief $P(f)$ about how likely various functions are, and our collection $\mathcal{F}$ of observations of the function, which give us insight into the specific f we are dealing with. We can use the accumulating knowledge about f to find the $x \in \mathcal{X}$ that solves our problem. More realistically, we can use the knowledge to help us approximately solve our problem, or perhaps just get ourselves headed in the right general direction. The problem as stated is very difficult.

Optimization methods that take this probabilistic approach are known as "Bayesian optimization" methods. These methods are unique in that they retain *all* accumulated data about the function and use all of it to determine where to search next.

1.1 Objective and Contributions

Our objective in this thesis is to make these methods more widely applicable and appealing to scientists who need global optimization techniques. There are three main obstacles preventing the wider adoption of Bayesian methods. First, the methods were never extensively tested and therefore not widely publicized, so the user community tended to be either unaware of such techniques or suspicious of their effectiveness. Second, there are many free parameters in Gaussian process-based optimization that are intimidating to users who are looking for a more black-box solution. Third, Bayesian methods do not make use of gradient information that is available in some problems, leading users to select other methods that can take advantage of such information.

We have made the following contributions that address these problems:

- We develop new GP-based techniques that are invariant to vertical shifting and scaling of the objective function. This reduces the need to tune the optimization algorithm's parameters to different objective functions.
- We develop a new methodology for evaluating global optimization techniques based on generating many (i.e. tens of thousands) test functions and evaluating performance on each. Our results show that GP-based optimization methods can perform well in a variety of situations.
- We show how using a maximum a-posteriori objective can be used to reliably learn the kernel parameters, eliminating the need for expensive pre-acquisition of the function and again reducing the need for user input.
- We present a novel prior based on the expected Euler characteristic of a Gaussian process.
- We give a polynomial time algorithm for computing the expected Euler characteristic of a useful subclass of Gaussian processes.
- We illustrate the use of gradient information with GP optimization, showing that the methods we have developed work well when given this information. Our empirical results show that GP-based optimization with gradient information can perform better than a quasi-Newton method with random restarts.

Chapter 2

Background

The study of Bayesian optimization methods draws on knowledge from the fields of optimization, machine learning, and random field theory. We begin by reviewing the contributions of each of these fields to the problem of global optimization.

2.1 Optimization in $\mathbb{R}^d$

As we discussed, the general problem of Equation 1.1 is far too broad for our purposes. To narrow things down, we will restrict ourselves to $\mathcal{X} \subset \mathbb{R}^d$ and $f : \mathcal{X} \to \mathbb{R}$. We will suppose that $\mathcal{X}$ is compact, and that f is Lipschitz-continuous, i.e. $\forall x, z \in \mathcal{X},\ |f(x) - f(z)| \leq l \cdot ||x - z||$ for some constant $l \geq 0$. This is sufficient to guarantee that a solution exists, i.e. $\exists x^* \in \mathcal{X}$ s.t. $f(x^*) = \sup_{x \in \mathcal{X}} f(x)$ [18]. However, even with these restrictions we cannot in general find the answer we seek using a finite number of function evaluations.

To remedy this, we could consider the problem successfully solved when we find a point x^* that we know is within ϵ of optimal in terms of Euclidean distance ($||x^* - x^*|| < \epsilon$) or function value ($|f(x^*) - f(x^*)| < \epsilon$) or some other closeness measure. This guarantees a solution using a finite number of function evaluations, but affords a theoretical comfort only: Any algorithm that provides this type of guarantee must make $\Omega(\left[\frac{l\sqrt{d}}{2\epsilon}\right]^d)$ function evaluations in the worst case [18], and we are interested in solving problems where d is at least in the dozens. Intuitively, this bound arises because we have to make sure we evaluate the function at points sufficiently near to all points in the domain, which means we have to construct something like a grid of points in d dimensions with a resolution determined by l and ϵ. The number of points in such a grid will scale exponentially in d.

2.1.1 Local Optimization

This insistence on finding a point that is near-optimal compared to *all other points in the domain* illustrates the critical difference between "global" optimization and "local" optimization: For the problem of local optimization, we seek a point x^+ such that

$$f(x^+) \geq f(x),\ \forall x \in \{x \in \mathcal{X} : ||x^+ - x|| < \epsilon\} \tag{2.1}$$

To verify that x^+ is an acceptable solution, we need only consider points in the neighbourhood of x^+. If f is differentiable, as is commonly supposed, this can be accomplished implicitly by examining first and second derivatives of f at x^+. This verification can be accomplished by factoring the (possibly approximated) Hessian matrix $H_f(x^+)$, which takes $\mathcal{O}(d^3)$ time. Therefore if we can *find* a candidate x^+ reasonably quickly, we can easily verify that we have found a solution to the local optimization problem. The business of getting to a good x^+ has been studied at least since the publication of Newton's method [32] approximately 300 years ago. Given a sufficiently close starting guess, the error in the solution given by Newton's method decreases geometrically after each step. Modern approaches that approximate Newton's method, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method discussed in Section 4.2, are efficient enough to solve local optimization problems with thousands of variables. In some lucky instances, notably when $f(x)$ is quasiconvex, a solution to the local problem is a solution to the global problem; if f is not quasiconvex, typically all bets are off. Nevertheless, local optimizers are frequently applied to non-quasiconvex problems in the hopes of finding a point that is at least better than a random (or educated) guess.

Realistic Optimization: The Principle of Perseverance We have seen that global optimization is hopelessly hard, and that local optimization is hopelessly easy. This is an exaggeration of course, but not far off the mark. However, the intractability of a problem does not make it go away, and the world is full of non-convex functions that various people would like to optimize even if they do not have the time to wait around for a provably optimal solution. This raises an important question when approaching non-convex global optimization problems: What if we have more than enough resources to find a local optimum, but not enough to guarantee finding a global optimum? This situation has become more prevalent with advances in computer hardware in the last forty years or so, and research on this problem has produced a motley crew of heuristic global optimization methods. We will

now briefly examine a variety of well-known global optimization techniques, including both non-Bayesian approaches and the Bayesian methods that will be the focus of this thesis.

2.1.2 Non-Bayesian Methods

The majority of global optimization algorithms that are widely known and used by scientists in different fields have little or no connection with Bayesian statistics. Most are based on intuitions about how to balance attraction toward a local minimum with prevention from getting stuck there.

Random Restarts

Since local optimization routines work so well, it seems natural to adapt them for use in global optimization. The simplest adaptation involves random restarts: Points are uniformly drawn from the set $\mathcal{X}$, and a local optimizer is executed starting from each one. As the number of samples approaches infinity, we will have found all possible local optima, and can simply pick the best one we have observed.

Typically, each local optimum x^+ has a *basin of attraction*—a set $\mathcal{B}(x^+) \subset \mathcal{X}$ such that for any $x \in \mathcal{B}(x^+)$ the optimizer when started from x will converge on x^+. Normally at least some of $\mathcal{B}(x^+)$ exists in the vicinity of x^+, so that many points near any given local optimum converge to the same location. This is a direct result of using Newton-like methods that assume the function is locally quadratic in a neighbourhood of x^+. "Density Clustering" [18] is a modification to the random restart approach that attempts to prevent the local optimizer from repeatedly falling into the same basin of attraction. The details are somewhat complicated, but the the basic idea is to build a region around each local optimum found so far and avoid starting the local optimizer within these regions. This is achieved by rejecting randomly sampled starting points that fall within these regions. This encourages the method to converge to new local optima as evaluation progresses.

Simulated Annealing

Another famous global optimization technique based on random sampling is known as simulated annealing. In this approach, a random walk is defined over the domain of the function. The hope is that, in the limit, the walk will converge to the global maximum of f. The walk is defined by two distributions: the *proposal* distribution, which is typically uniform

over $\mathcal{X}$, and the *acceptance* distribution, given by

$$P(x_{\text{curr}} \leftarrow x_{\text{prop}}) = \min\{1, e^{(f(x_{\text{prop}}) - f(x_{\text{curr}}))/T}\} \tag{2.2}$$

The procedure is simple: Propose a point x_{prop}. Evaluate the function at that point. Assign $x_{\text{curr}} \leftarrow x_{\text{prop}}$, the "current point", according to the acceptance probability above. One can see that if $f(x_{\text{prop}}) \geq f(x_{\text{curr}})$, we always move to the newly proposed point, otherwise we move to it with some probability dependent on the magnitude of the decrease in function value and on the current "temperature" T. Initially, T is taken to be large; in this situation we will accept almost any point regardless of how poor it is. As we decrease T, we will accept inferior points with decreasing probability. It is possible to show that, under certain conditions, as $T \to 0$ and the number of proposals goes to infinity, x_{curr} converges to x^* [18]. In practice, since this convergence may take an exceedingly long time, we keep track of the best function value seen to date, and stop the walk once our time runs out.

Lipschitz Methods

We now turn our attention to methods that are entirely deterministic. Previously, we mentioned that we will assume the functions we are optimizing are Lipschitz-continuous, meaning there exists a constant l for which

$$\forall x, z \in \mathcal{X},\ |f(x) - f(z)| \leq l \cdot ||x - z|| \tag{2.3}$$

This property effectively gives an upper and lower bound on the function that is refined as more and more function values are observed. These bounds are in turn used to decide which observation to acquire next. There are many variations, but a common approach is to evaluate f where its upper bound is greatest. After acquiring this new function value, the upper bound is updated and maximized again to find the next point to acquire. This technique can provide a provably ϵ-optimal solution, but requires $\Omega(\left[\frac{l\sqrt{d}}{2\epsilon}\right]^d)$ in the worst case, as mentioned earlier.

Homotopy Methods

Homotopy methods take a significantly different approach. They attempt to take a solution to a simplified version of the problem and use that to find solutions to the original problem.

A *homotopy* H is a differentiable map that "blends" two functions.

$$H : \mathbb{R}^d \times [0,1] \to \mathbb{R} \tag{2.4}$$

$$H(x,0) = g(x) \tag{2.5}$$

$$H(x,1) = f(x) \tag{2.6}$$

For example, $H(x,t) = (1-t) \cdot g(x) + t \cdot f(x)$ is a homotopy that varies smoothly with the parameter t from g to f. The premise of these methods is to construct a homotopy for which the function g is easy to optimize. Since the homotopy is a continuous map, the location of a local optimum usually changes smoothly as we change t. We can therefore find the optima of the easy function $H(x,0)$, and set up differential equations to track the optima as we move t from 0 to 1. This works well in some instances; however, local minima can undergo bifurcation and merging, and they can spontaneously appear or disappear as the parameter t changes. Care must be taken in each of these situations. Another strange possibility is the "turn-back." There may be a continuous path through $\mathbb{R}^d \times [0,1]$ connecting an optimum of g with an optimum of f that is *not monotonic in* t. That is, we may have to increase t for a time, then backtrack t but continue moving x to a different part of the space, and then increase t again to reach the optimum of f at the other end of the path. Of course, this can happen several times depending on the objective and the homotopy used. Reliably tracking solutions in this way has resulted in interesting algorithms for optimization that are unlike any other known techniques.

Genetic Algorithms

The term "genetic algorithm" is applied to a great swath of algorithms that share a common metaphor. In this paradigm, a "population" of "individuals", described by their "chromosomes", are "evolved" over time according to their "fitness." That is, a set of candidate solutions is maintained, these candidates are perturbed according to various biologically-inspired rules, and the resulting new candidates are evaluated using f. Those individuals that perform poorly are (probably) removed from the population. This procedure repeats until no further progress is made.

Genetic algorithms present a staggering array of choices governing the details of the optimization procedure. Good choices for the specific rates and mechanisms for "crossover", where two solutions are combined, and "mutation", where a solution is perturbed by an outside force, are critical for good performance. In addition, since their original description by

Holland [15], genetic algorithms have acquired innumerable additional tricks and heuristics that help in specific situations.

Ultimately there is no single, unified "genetic algorithm". The application of genetic algorithms is a high level technique of taking the popular perception of evolution and natural selection and constructing an algorithm whose functions and data structures, according to the developer, mimic those present in nature. This in effect means that, to use a genetic algorithm effectively, a great deal of expert knowledge is needed both about the function in question *and* about genetic algorithms. This is because genetic algorithms are procedural rather than constructive—knowledge about the objective (smoothness, range, etc.) must be translated into a genetic algorithm that will likely work well on such functions.

2.1.3 Introduction to the Bayesian Approach to Optimization

Next, we introduce the details of "Bayesian" optimization methods. In contrast to the non-Bayesian methods described earlier, which are focussed predominately on algorithm development, Bayesian optimization methods take a different approach: They combine relatively simple algorithms with explicit, descriptive statistical models of the objective function. This gives Bayesian methods a major advantage, because a practitioner can improve performance by describing the *objective function* well, as opposed to describing a *solution algorithm* well, which is much more difficult for a non-expert.

The adjective *Bayesian* can be applied to any optimization technique that makes use of the laws of probability to combine prior belief with observed data to compute a posterior distribution over any quantity of interest. For most methods that fall into the category of Bayesian optimization, the quantity of interest is the function itself: These methods model the function in question by encoding prior beliefs about the function, updating those beliefs according to the laws of probability as new information about the function accumulates, and using the resulting model to guide the progress of the optimization procedure. This model is sometimes called a "surrogate" function, or a "response-surface."

This does not restrict Bayesian methods to those that model the objective explicitly; one could imagine that modeling a different quantity (such as the location of the function's optimum) could give rise to effective optimization procedures. Nevertheless, the methods we discuss here will take the approach of building a response-surface model that has been augmented with the capability of probabilistic reasoning.

Models and Acquisition Criteria A Bayesian response-surface method for optimization has two major components: The probabilistic *model* describing the objective function, and the *acquisition criterion*[1] function that determines how to choose the next point at which we evaluate the function.

The *Gaussian process* model is used in all of the Bayesian response-surface methods we will discuss here, in one guise or another. We introduce the basic Gaussian process now to provide the appropriate background material, but we will discuss enhancements to the model that are widely used by the machine learning community in Section 2.2. In principle, there is no reason we could not use other, more complex models; it could be that in some instances a more general exponential family model, say that can model higher-order interactions (beyond pairwise) may be useful. The major problem, as usual, is computational: more general models will not admit the clean, analytical inference procedures that Gaussian processes afford. Nonetheless, they would be an interesting alternative avenue of research.

2.1.4 Preview of Gaussian Processes

A *Gaussian process* (GP) [46, 35] is a collection[2] of random variables $\{F_{x^1}, F_{x^2}, ...\}$ for which any finite subset of the variables has a joint multivariate Gaussian distribution. The variables are indexed by elements x of a set $\mathcal{X}$. We will restrict our attention to $\mathcal{X} \subset \mathbb{R}^d$, but this is not necessary; $\mathcal{X}$ could be the space of integers or trees or strings, for example. For any finite length vector of indices $\mathbf{x} = [x^1, x^2, ..., x^n]^T$, we have a corresponding vector $\mathbf{F_x} = [F_{x^1}, F_{x^2}, ..., F_{x^n}]^T$ of variables that have a joint multivariate Gaussian (or *normal*) distribution,

$$\mathbf{F_x} \sim \mathcal{N}\left\{\mu_0(\mathbf{x}), k(\mathbf{x}, \mathbf{x})\right\}, \tag{2.7}$$

where the elements of $\mu_0(\mathbf{x})$ are given by a prior mean function $\mu_0(x_i)$, and k is the *kernel* function. The kernel takes two indices x^i and x^j, and gives the covariance between their corresponding variables F_{x^i} and F_{x^j}. Given vectors of indices $\mathbf{x}$ and $\mathbf{z}$, k returns the matrix of covariances between all pairs of variables where the first in the pair comes from F_{x^i} and the second from F_{z^j}. The result of $k(\mathbf{x}, \mathbf{x})$ must be a square, symmetric positive semi-definite matrix for any $\mathbf{x}$ in order for k to be a valid kernel [42]. Note that each F_{x^i} is marginally Gaussian, with mean $\mu_0(x^i)$ and variance $k(x^i, x^i)$.

[1] This function has been by various names in previous work, including the 'acquisition criterion' [39].'

[2] All of our GPs will have an uncountable number of variables, but we will abuse notation and "list" them occasionally, though this is impossible.

Gaussian Process Regression Suppose we have a function $f(x)$ that we would like to model. Furthermore, suppose that we cannot observe f directly, but that we can observe a random variable F_x that is indexed by the domain of f and whose *expected value* is f, i.e., $\forall x \in \mathcal{X},\ \mathrm{E}[F_x] = f(x)$. In particular, we assume that our prior belief about the function f conforms to a Gaussian process with prior mean μ_0 and kernel k, as described above, and that the observed variable F_x is an observation of $f(x)$ that has been corrupted by zero-mean, i.i.d. Gaussian noise, i.e., $F_x = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Hence, $f(x)$ is a hidden variable whose posterior distribution we can infer after observing samples of F_x at various locations in the domain. The resulting inference is called Gaussian process regression.

Let $\mathbf{x}$ be the set of observation points and $\mathbf{f}$ be the corresponding real-valued observations. We want to compute the posterior distribution of some new point $z \in \mathcal{X}$. This distribution will be Gaussian with mean and variance given by

$$\mu(F_z|F_\mathbf{x} = \mathbf{f}) = \mu_0(z) + k(z,\mathbf{x})(k(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathrm{I})^{-1}(\mathbf{f} - \mu_0(\mathbf{x})) \tag{2.8}$$

$$\sigma^2(F_z|F_\mathbf{x} = \mathbf{f}) = k(z,z) - k(z,\mathbf{x})(k(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathrm{I})^{-1}k(\mathbf{x},z). \tag{2.9}$$

Note that the inverse applies to the kernel matrix of observed domain points (plus a diagonal term), and so can be computed once and used to evaluate the posterior at many points in the domain. (In practice, the matrix will be Cholesky factored instead of inverted.)

Gaussian process regression is a generalization of least squares linear regression that allows for more complex regression functions, and provides information about the uncertainty of the regression model at different domain points[3] Use of this type of model is known as "kriging" [4] in geostatistics[4]. The form of the possible regression functions is not as complex as one might initially suspect: Consider the posterior mean function $\mu(F_z|\mathbf{F}_\mathbf{x} = \mathbf{f})$. It consists of the prior mean function $\mu_0(z)$, which is frequently taken to be constant, plus a term involving the kernel function between the query point z and each observed data point. Since the second part of the second term, $(k(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathrm{I})^{-1}(\mathbf{f} - \mu_0(\mathbf{x}))$, depends only on the observed data, this factor collapses to a simple weight vector independent of z: The posterior mean is just a weighted sum of kernel functions between each observed data point and the query point z. In other words, the posterior mean function is a linear combination of n kernel functions, each one centered at an observed data point. From this construction,

[3] For $\mathcal{X} \subset \mathbb{R}^k$ and $k(x_i, x_j) = x_i \cdot x_j$, this formulation is equivalent to linear ridge regression with regularizer σ_ϵ^2.

[4] Actually "kriging" refers to a model that is the sum of a low-degree polynomial and a Gaussian process. The polynomial portion is typically taken to be constant, so in most cases the resulting models are identical.

one can see that the shape of the mean function and of the functions a given GP is likely to produce is governed largely by k, the kernel function.

Introduction to Acquisition Criteria Now that we have a model, we would like to make use of it to decide where to acquire data about the objective in order to find its optimum using as few function evaluations as possible. Since the model provides a distribution over the value of the objective at every domain point, we can search through the model for a good point to acquire next, either because the function is likely to be near-optimal at that point, or because an observation there will provide a large amount of information about the shape of the function, or because the resulting observation will have some combination of these properties. This idea will be formalized as we examine the history of Bayesian methods.

2.1.5 A History of Bayesian Optimization Methods

Bayesian optimization approaches have existed in the scientific literature for about forty years. Their capability and complexity have increased in response to the explosion in available computing power over that time period as they evolved into the methods in use today. For the remainder of this section, we will assume we wish to maximize an objective function $f : \mathbb{R}^d \to \mathbb{R}$.

Kushner

One of earliest publications in English that describes a Bayesian response-surface approach to optimization was written by Harold J. Kushner in 1964 [25]. Kushner describes a method for optimizing a one-dimensional, real-valued function.

Model Kushner's work uses a specific type of Gaussian process known as a Wiener process to model the unknown function f, and assumes the observed variable F_x is the underlying function plus i.i.d. Gaussian noise, i.e.

$$F_x = W_x + \epsilon \tag{2.10}$$

$$\epsilon \sim \mathcal{N}(0, \sigma_n^2) \tag{2.11}$$

The Wiener process in one dimension describes the Brownian motion random walk, and can be simply characterized. Suppose $a < b < c < d \in \mathbb{R}^+$. Then we have

$$W_0 = 0 \tag{2.12}$$

$$W_b - W_a \sim \mathcal{N}(0, \sigma_f^2 \cdot (b-a)) \tag{2.13}$$

$$[a,b) \cap [c,d) = \emptyset \implies (W_a - W_b) \text{ and } (W_c - W_d) \text{ are independent} \tag{2.14}$$

Functions sampled from such a process are almost surely continuous, but nowhere differentiable. Because of the *independent increments* property (2.14), the posterior probability distribution at any point depends only on the two closest observed data points. We therefore do not need the full-blown machinery of Gaussian processes described in Section 2.1.4 for inference, since the posterior at any point will depend on at most two of our observed data points. The parameter σ_f^2 controls how quickly the variance grows as we move away from observed data, which affects the amplitude of the larger scale function variation we are likely to observe. Suppose we want the posterior distribution of F_z, and that we have data points (x^i, f^i) for $x^1 < x^2 < ... < x^n$. If $x^i < z < x^{i+1}$, then

$$F_z \sim \mathcal{N}(\mu(F_z|\mathbf{F_x} = \mathbf{f}), \sigma^2(F_z|\mathbf{F_x} = \mathbf{f})) \tag{2.15}$$

$$\mu(F_z|F_\mathbf{x} = \mathbf{f}) = \frac{(\sigma_n^2 + \sigma_f^2 \cdot (x^{i+1} - z))f_{x^i} + (\sigma_n^2 + \sigma_f^2 \cdot (z - x^i))f_{x^{i+1}}}{2\sigma_n^2 + \sigma_f^2 \cdot (x^{i+1} - x^i)} \tag{2.16}$$

$$\sigma^2(F_z|F_\mathbf{x} = \mathbf{f}) = \frac{(\sigma_f^2)^2 \cdot (z - x^i)(x^{i+1} - z) + \sigma_n^2 \cdot (\sigma_n^2 + \sigma_f^2 \cdot (x^{i+1} - x^i))}{2\sigma_n^2 + \sigma_f^2 \cdot (x^{i+1} - x^i)} \tag{2.17}$$

Note that for $x^i < z < x^{i+1}$, $\mu(F_z|F_\mathbf{x} = \mathbf{f})$ is linear in z, and $\sigma^2(F_z|F_\mathbf{x} = \mathbf{f})$ is quadratic in z. For $x^1 < z < x^n$, both are continuous, $\mu(F_z|F_\mathbf{x} = \mathbf{f})$ is piecewise linear and $\sigma^2(F_z|F_\mathbf{x} = \mathbf{f})$ is piecewise quadratic[5]. They are differentiable except at the data points $\{x^i\}$. In the zero noise case where $\sigma_n^2 = 0$, $\mu(F_z|F_\mathbf{x} = \mathbf{f})$ interpolates the data points.

[5]Outside the range of the data points they become constant and linear respectively; the forms are similar to those given here.

Acquisition Criterion The criterion used by Kushner to select the next acquisition is given by

$$x^{n+1} = \operatorname*{argmax}_z P[F_z \geq \max_i(f_{x^i}) + \xi(n)] \tag{2.18}$$

$$= \operatorname*{argmax}_z \left\{ 1 - \Phi\left(\frac{\max_i(f_{x^i}) - \mu(F_z|F_{\mathbf{x}} = \mathbf{f}) + \xi(n)}{\sqrt{\sigma^2(F_z|F_{\mathbf{x}} = \mathbf{f})}} \right) \right\} \tag{2.19}$$

$$= \operatorname*{argmin}_x \left(\frac{\max_i(f_{x^i}) - \mu(F_z|F_{\mathbf{x}} = \mathbf{f}) + \xi(n)}{\sqrt{\sigma^2(F_z|F_{\mathbf{x}} = \mathbf{f})}} \right) \tag{2.20}$$

$$= \operatorname*{argmin}_x \left(\frac{(\max_i(f_{x^i}) - \mu(F_z|F_{\mathbf{x}} = \mathbf{f}) + \xi(n))^2}{\sigma^2(F_z|F_{\mathbf{x}} = \mathbf{f})} \right) \tag{2.21}$$

where Φ is the standard normal cumulative density function, which is monotonic in its argument; hence the equivalence of (2.19) and (2.21). This criterion, which we call Maximum Probability of Improvement (MPI), is the posterior probability that the next acquisition observed $f(x^{n+1})$ is greater than the current maximum observed value plus a positive term $\xi(n)$. The parameter $\xi(n)$ is used to control how "local" or "global" the search is, and is typically allowed to depend on n, the number of data points we have so far. As $\xi(n) \to \infty$, using (2.21) as a acquisition criterion degenerates to choosing the point of highest posterior variance. Conversely, as $\xi(n) \to 0$, the criterion selects the point with highest posterior mean. Any positive, bounded $\xi(n)$ results in a tradeoff between mean and variance. Once $\xi(n)$ is chosen, the minimum in (2.21) can be computed analytically on each interval $[x^i, x^{i+1})$ using the equations above, and the best of these can be chosen as the next acquisition. It therefore takes time linear in the number of data points to choose the next acquisition location.

The question of how to choose $\xi(n)$ is similar to the question of how to choose temperature in simulated annealing: Kushner suggests starting $\xi(n)$ quite high (based on some intuition about the function) and scheduling its decrease in a geometric fashion as data accumulates. He suggests accomplishing this by manually defining regions of interest and desired progress for each stage; however, no concrete algorithm is given in the work.

GROPE

This method, published by Elder in 1992 [11], is one representative of the various heuristics [33, 44] used to extend Kushner's original method to multiple dimensions.

Model The model used by Elder is inspired by the Brownian motion model used by Kushner. The convex hull of the points that have been acquired is divided into simplexes

using a Delaunay triangulation [43]. The posterior mean along edges of the triangulation is constrained to be the linear interpolator between the points, and the posterior variance is constrained to be quadratic. This is in effect assuming that a Wiener process runs along the edges of the triangulation. Using the distributions defined along each edge, a posterior mean and variance is defined over each simplex, again constrained to be linear and quadratic respectively, but in d dimensions. This choice is intended to emulate the Markov property of the Wiener process, where the posterior distribution depends only on the "nearest" points, since the distribution of a point depends only on the vertices of its enclosing triangle. The use of a triangulation in this manner, however, is at best an approximation to this property: Triangles that have one very large angle (i.e. that are very skinny) will have points along their edges whose posterior distribution depends not on the closest point (the vertex opposite the edge) but on points potentially much further away (the vertices incident to that edge). It is hoped that the use of the Delaunay triangulation will mitigate this problem, since it prefers more equi-angular triangles.

Acquisition Criterion Following Kushner's example, Elder uses the same criterion

$$x^{n+1} = \underset{x}{\operatorname{argmin}} \left(\frac{(\max_i(f_{x^i}) - \mu(F_x|F_{\mathbf{x}} = \mathbf{f}) + \xi(n))^2}{\sigma^2(F_x|F_{\mathbf{x}} = \mathbf{f})} \right) \tag{2.22}$$

to choose which data to acquire next. This minimization problem is broken into subproblems over each simplex: A numerical optimizer is used to find the best point within each simplex, and these are compared to find the best overall candidate.

Elder [11] claims that

> The Key difficulty in expanding Kushner's algorithm from $\mathbb{R}^1$ to $\mathbb{R}^d$—and perhaps the reason the method saw little use for a generation—is the extension of the random walk model into a random *field* (for which there are even competing theoretical definitions in the literature).

There are several well-known Gaussian fields whose sample paths satisfy his desiderata of being "locally rough" but "regionally smooth"; furthermore, these processes were suggested much earlier as possible candidates for use in higher-dimensional global optimization problems. The main drawback is not in the conceptual definition of random fields; rather it is in the time complexity of reasoning with them: typically operations take time quadratic or cubic in the number of data points. We will examine this difficulty further in Section 2.2.

Mockus

At the time Kushner was originally developing his algorithm, Jonas Mockus was in the former USSR developing similar methods and publishing in several Russian language journals of the day. The earliest of his English language publications that Mockus cites in his book is in the proceedings of the International Federation for Information Processing Congress 1977 which was held in Toronto, Canada.

Model Mockus suggests a model that is an extension of the Wiener process to d dimensions. This model was chosen because it is, in his opinion, the simplest that satisfies two desiderata: Sampled functions are continuous, and their finite differences are independent. Finite differences are the discrete analogues of derivatives; again functions sampled from this model are nowhere differentiable. The kernel function for this model is given by

$$k(x,z) = \sigma_f^2 \prod_{i=1}^{d} (1 - \frac{x_i - z_i}{2}) \tag{2.23}$$

where d is the dimension of the data points, and $x, z \in [-1,1]^d$ for simplicity[6]. (Any hyper-rectangular region is easily mapped to $[-1,1]^d$.) This model is the sum of 2^d Wiener *fields*, each of which is the extension of the Wiener *process* to d dimensions. The Wiener fields have their origins at each vertex of $[-1,1]^d$. The result is a process in d dimensions that is everywhere continuous but nowhere differentiable.

Acquisition Criterion The development of Mockus's criterion begins from assuming we will have the opportunity to observe n samples, and that we wish to minimize the expected difference between the function value we report and the true optimum value of the function. Given this criterion, the rational point to report is the point that gives the lowest expected loss, according to our posterior model. We call the reported point x^{n+1}.

$$\begin{aligned} x^{n+1} &= \operatorname*{argmin}_z \mathrm{E}[(f^* - F_z)|f(\mathbf{x})] && (2.24) \\ &= \operatorname*{argmin}_z (\mathrm{E}[f^*] - \mathrm{E}[F_z|f(\mathbf{x})]) && (2.25) \\ &= \operatorname*{argmax}_z \mathrm{E}[F_z|f(\mathbf{x})] && (2.26) \end{aligned}$$

that is, the point with the highest posterior expected value. In practice, the reported point is chosen from the set of points where we have observed F instead of over the entire domain. For the Wiener process and extensions with no added noise the two choices are equivalent,

[6] Recall that x_i is the ith component of the point $x \in \mathbb{R}^d$.

but for more general models this is not the case. This simplification is made for two reasons: It is computationally much more convenient to simply scan the list of observed values for the best one than to search over $\mathcal{X}$, and there is typically a preference for reporting a point that has low posterior variance, which our observed points have—a preference that is *not* stated in the original problem. Nevertheless, for the moment we will presume that we will always report the point with the greatest posterior expected value.

Having decided which point to report if we cannot acquire any more, we now examine the optimal point to observe if we are allowed one more acquisition, which leads us to what Mockus calls the "one-stage" method.

$$x^n \leftarrow \operatorname*{argmin}_{x^n} \mathrm{E}[\ \min_{x^{n+1}} \mathrm{E}[f(x^*) - F_{x^{n+1}} | f_{x^{1:n}}] \mid f_{x^{1:n-1}}\ ] \tag{2.27}$$

$$\leftarrow \operatorname*{argmax}_{x^n} \mathrm{E}[\ \max_{x^{n+1}} \mathrm{E}[F_{x^{n+1}} | f_{x^{1:n}}] \mid f_{x^{1:n-1}}\ ] \tag{2.28}$$

The property we want x^n to have is intuitive: Choose x^n so that after we have observed F_{x^n}, the maximum posterior mean of $F_{x^{n+1}}$, the point we will report, is as large as possible, *given our actual observation of* F_{x^n}.

Of course in principle there is no reason not to consider all future acquisition; we can unroll the recurrence relation all the way down to the first acquisition:

$$x^1 \leftarrow \operatorname*{argmax}_{x^1} \mathrm{E}[\ \max_{x^2} \mathrm{E}[\ \ldots\ \max_{x^n} \mathrm{E}[\ \max_{x^{n+1}} \mathrm{E}[F_{x^{n+1}} | f_{x^{1:n}}] \mid f_{x^{1:n-1}}\ ] \ldots \mid f_{x^1}]\ ] \tag{2.29}$$

Thus to really maximize our criterion from the beginning of an optimization run, we must consider all possible future trajectories of the choices that we make and the values we might observe. This "expecti-max" quantity appears most famously in the problem of acting optimally in finite-horizon Markov Decision Processes as the Bellman equation, and various attempts have been made to approximate its solution [45, 20]. However, to our knowledge, all current Bayesian response-surface methods with an expected-loss type criterion use only the "one-stage" method for selecting the next point. Mockus re-writes the criterion thus

$$x^n \leftarrow \operatorname*{argmax}_{x^n} \mathrm{E}[\ \max_{x^{n+1}} \mathrm{E}[F_{x^{n+1}} | f_{x^{1:n}}] \mid f_{x^{1:n-1}}\ ] \tag{2.30}$$

$$\leftarrow \operatorname*{argmax}_{x_n} \mathrm{E}[(\max_{z} \mathrm{E}[F_z | f_{x^{1:n}}] - \max_{w} \mathrm{E}[F_w | f_{x^{1:n-1}}])_+] \tag{2.31}$$

$$\leftarrow \operatorname*{argmax}_{x^n} \mathrm{E}[(F_{x^n} - \mu_{\max})_+] \tag{2.32}$$

which is the expected positive difference between the value we would have reported *before* observing $f(x^n)$ and *after* observing $f(x^n)$. This re-writing is based on the assumption that observing a new point will never cause us to report a value that is worse than we would have

reported before, and that we will only report a function value at a point we have observed. This means that $\max_w \mathrm{E}[F_w | f_{x^{1:n-1}}]$ is a constant (which we write as $\mu_{\max}$) and the whole criterion is a single expectation of a function of F_{x^n}. This criterion is frequently called *expected improvement*. Furthermore, Mockus augments this criterion with a parameter ξ similar to that used in MPI, giving

$$x^n \leftarrow \operatorname*{argmax}_{x^n} \mathrm{E}[(F_{x^n} - (\mu_{\max} + \xi))_+] \tag{2.33}$$

For the remainder of this thesis, we will refer to this as the Maximum Expected Improvement (MEI) criterion.

EGO

Jones, Schonlau and Welch describe an algorithm they call Efficient Global Operation or EGO [19].

Model The stochastic model used for EGO is known as the Design and Analysis of Computer Experiments or DACE model [37, 38]. This model was used by Sacks et al. for a different task—choosing acquisitions for an experimental design that minimizes a global error measure, such as expected squared error between the model and the true function integrated over the domain. The DACE model is a Gaussian process model with a parameterized kernel

$$k(x, z) = \sigma_f^2 \cdot \mathrm{e}^{-\sum_{\ell=1}^{d} \theta_\ell |x_\ell - z_\ell|^{p_\ell}} \tag{2.34}$$

where $\theta_\ell \geq 0$ and each $p_\ell \in [1, 2]$. This is a generalization of the exponentiated-negative-distance kernel commonly used by the machine learning community. Covariance along each axis is controlled by the positive length-scales θ_ℓ, and by the exponents p_ℓ. If $p_\ell = 2$, sample functions will be infinitely differentiable along the direction of x_ℓ; otherwise they will be non-differentiable but continuous.

Acquisition Criterion The criterion used for selecting query points in EGO is exactly the "one-stage" or "expected improvement" criterion used by Mockus. The developers of EGO note that this quantity can be computed using standard Gaussian probability and density functions. For the remainder of this section, we will use $\mu(x)$ and $\sigma(x)$ as shorthand for the posterior mean and standard deviation of F_x, we will use $f_{\max}$ as shorthand for

$\max_i f_{x^i}$, and the functions Φ and ϕ will be the standard normal CDF and PDF.

$$\mathrm{E}[(F_x - f_{\max})_+] = (\mu(x) - f_{\max}) \cdot \Phi\left(\frac{\mu(x) - f_{\max}}{\sigma(x)}\right) + \sigma(x) \cdot \phi\left(\frac{\mu(x) - f_{\max}}{\sigma(x)}\right) \quad (2.35)$$

Before following this criterion, however, the EGO algorithm first acquires approximately $10 \cdot d$ points in a Latin hypercube design where all one- and two-dimensional projections are nearly uniformly covered. The DACE model is then fit to these points using maximum likelihood. If the fit is found by inspection to be "poor", the data are transformed using a log or inverse $(-1/y)$ transformation. The criterion function just described is then optimized using a branch-and-bound technique to find the next point to acquire. This procedure is followed until the maximum expected improvement found is less than 1% of the current best function value.

The EGO procedure takes basically the same approach as Mockus's work except that it uses a more complex model and provides a systematic way of optimizing the criterion function used for selecting acquisition points. Its practical use has been restricted to low-dimensional ($d \leq 6$) spaces to date, however, partly because use of the initial Latin hypercube can be undesirable for expensive-to-evaluate problems,[7] and because the branch-and-bound algorithm alluded to does not scale up effectively to higher dimensions.

Extensions to EGO

Extensions to the EGO procedure have been introduced that enable further control of the locality of search points, allow constraints on other response variables, and reduce computation time for problems that have mixture of expensive and cheap objectives and constraints [41, 39].

Generalized Expected Improvement Schonlau [41] gives a recurrence relation for computing the expectation of positive integer powers of improvement.

$$\mathrm{E}[((F_x - f_{\max})^g)_+] = \sigma(x)^g \sum_{k=0}^{g} (-1)^k \left(\frac{g!}{k!(g-k)!}\right) z^{g-k} T^k \quad (2.36)$$

where

$$T^k = -\phi(z) \cdot z^{k-1} + (k-1) \cdot T^{k-2} \quad (2.37)$$

and

$$z = \frac{f_{\max} - \mu(x)}{\sigma(x)} \quad (2.38)$$

[7] We have found that for AIBO walk optimization, which has 15 dimensions, we can find a very good walk in $150 = 10 \cdot d$ evaluations by following the maximum expected improvement criterion. In this case, it is unlikely a Latin hypercube design would fare as well.

The parameter g provides another mechanism for controlling the locality of the search procedure: By maximizing the expected squared improvement (or cubed improvement or...) preference is shifted from points that have a low posterior mean to points that have a high posterior variance. This is similar to the tradeoff afforded by changing ξ discussed earlier. Schonlau suggests that, since expected improvement is frequently found to be "too local", setting $g = 2$ gives better performance; however, this is based on one example only. Sasena gives a "cooling" schedule for g that he has found to work well empirically.

Simultaneous Acquisitions The methods described up to this point have concerned themselves with acquiring points one at a time, always using the most current model of the function. There are situations, however, where acquiring several points simultaneously is preferable; if we have the resources to acquire points from an expensive function many times in parallel it seems a waste not to take advantage of this capability.

Suppose we want to acquire a group of m points. A simple extension of the expected improvement criterion would be

$$\mathrm{E}[\max(0, F_{x^1} - f_{\max}, F_{x^2} - f_{\max}, \ldots, F_{x^m} - f_{\max})] \tag{2.39}$$

While this definition is simple, the optimization problem it poses is potentially very hard. We must now solve a global optimization problem in $\mathbb{R}^{m \cdot d}$ since we need to select the location of m points. Also, it is not known if the expectation (2.39) has a closed form, so sampling is currently the only method for evaluating it.

For these reasons, Schonlau [41] suggests a simpler alternative: Use the one-step expected improvement criterion to find a point. Do not acquire the function's value at that point, but update the posterior variance assuming we have. (Note that the posterior variance depends only on the location of the acquisition, not on the observed value.) Repeat until we have chosen m points and then acquire them, possibly in parallel. This technique could be used with any of the other acquisition criteria also. Schonlau reports that it behaves reasonably on an example function using the expected squared improvement criterion.

Constraints Various extensions that allow more complex constraints on variables have been developed for use with EGO. Here we mention four of the most commonly used proposals.

The first two ideas are re-inventions of the most basic way of dealing with a constrained optimization problem: Penalize constraint violations by including them in the acquisition

criterion, and solve the resulting unconstrained problem. Constraints are presumed to be expressed in the form

$$\bigwedge_{i=1}^{k} g_i(x) \leq 0 \tag{2.40}$$

In the first approach, proposed by Schonlau, all constraint functions are modeled just as the objective function is, and the acquisition criterion (typically expected improvement) is multiplied by the posterior probability that all constraints are satisfied at any given point. This way, points will only be selected by the acquisition criterion if they are unlikely to violate the constraints. The constraint and objective variables are assumed to be independent for ease of modeling and computation. This method of penalizing will alter the location of optima of the acquisition criterion if there is any uncertainty in the model of the constraints. Worse, the penalized criterion can be very flat in areas where constraints are violated with very high probability, since the modified acquisition criterion is close to zero in these regions. This makes the already difficult problem of maximizing the acquisition criterion even harder.

The second penalty-based approach, suggested by Björkman and Holmström, is simply to add the value of the constraint functions $g_i(x)$ anywhere they are positive, i.e.

$$c^*(x) = c(x) + \sum_{i=1}^{k} [g_i(x)]_+ \tag{2.41}$$

If the g_i are expensive or not readily available, then they are modeled using a GP and the model's posterior mean is used instead. This additive method of penalizing does not affect the locations of feasible solutions, but it does introduce regions of non-differentiability wherever $\exists i \; g_i(x) = 0$. Any optimization routine used with this type of penalty must therefore be capable of handling non-differentiable functions.

A third approach defined by Audet et al. [3] involves randomly generating candidate points in the domain, and ranking them according to their *expected violation*. This quantity is exactly analogous to the expected improvement criterion used on the objective. As a penalty function, expected violation was found by Sasena to perform poorly on the example he presents. Constraints by definition are binary objective: they are violated or not. A point that with very high probability violates a constraint by a tiny amount (and therefore has a tiny expected violation) is not a feasible solution; if such a point is deemed acceptable then the problem has been incorrectly described.

The fourth approach acknowledges that decades of research in the field of constrained optimization have resulted in methods that are widely used, well understood, and very effec-

tive. Rather than apply a primitive penalty function for constraint violation, the posterior mean of the constraints (i.e. our best guess) can be handed off along with the acquisition criterion to any constrained optimization routine; a log barrier method would be a popular choice, for example.

Heterogeneous Function Costs Sasena [39] points out that it certain problems may have a mixture of objective and constraint functions with varying costs for evaluation. He proposes dividing these into two groups, "expensive" and "cheap", and treating them separately during optimization. The approach is an obvious one: If a function is expensive, model it with a GP. If it is not, simply respect the constraint while searching for the next acquisition point. If the *objective* is cheap, add the following constraint to the global optimizer used for the acquisition criterion:

$$f_{\text{cheap}}(x) \geq f_{\max} \tag{2.42}$$

This takes care of the situation where we want to test expensive constraints as little as possible by avoiding points we know cannot improve on what we have found so far. If everything is cheap, this approach, which Sasena calls "superEGO," degenerates to using the global optimizer chosen for the acquisition criterion. In Sasena's work, the DIRECT algorithm [12] is used.

Boyle

At the time of this writing, the most recent extensions to Gaussian process optimization were the addition of techniques prevalent in the Bayesian machine learning field by Boyle [6].

Model Throughout his dissertation, Boyle focuses on using a more fully Bayesian treatment of Gaussian processes. To this end, he specifies priors on covariance function parameters such as length scales, data rotations, and noise levels. He then uses Monte Carlo methods to infer the posterior, which is now no longer Gaussian. This allows him to use a more complicated covariance function, which we now detail.

The covariance function used by Boyle has a squared exponential form with a length scale θ_ℓ for each principal axis $\ell \in \{1, 2, ..., d\}$ and a fully parameterized rigid rotation matrix Λ. The form of the covariance function is

$$k(x, z) = \sigma_f^2 \cdot \mathrm{e}^{-(x-z)^T \Lambda [\mathrm{diag}(\theta_\ell^2)] \Lambda^T (x-z)} \tag{2.43}$$

where $[\text{diag}(\theta_\ell^2)]$ is a matrix with squared length scales on the diagonal and zeros elsewhere, and Λ is $d \times d$ with $\Lambda^T\Lambda = \mathrm{I}$. Boyle parameterizes the matrix Λ using a set of d^2 Givens rotation angles $-\pi < \rho_{ij} < \pi$. This kernel therefore has $d^2 + d + 1$ parameters that are used to achieve data rotations, data scaling, and function scaling by σ_f.

Using such a rich parameterization with a small amount of data will almost certainly lead to overfitting if we use a MAP hypothesis [13]. Therefore, rather than choosing a single set of kernel parameters, Boyle specifies prior beliefs about the parameters and integrates over them to compute the posterior of F. This cannot be done analytically, so Monte Carlo methods are used to approximate the posterior. Taking this approach allows the use of richer models, but is computationally much more expensive than using a simpler model with the maximum likelihood or MAP parameter settings.

Further to this idea, Boyle presents results where after each acquisition, two models are built: One is axis-aligned, and one is rotated. Posterior inferences are then constructed by weighting each model according to its posterior plausibility.

Finally, in an effort to curb computational costs, Boyle points out that it is possible to use Reduced Rank Gaussian Processes [36] (RRGPs) for optimization, and gives one empirical example. This approach is one of a set of techniques for reducing the computational cost of GP inference; we propose a more rigorous investigation of these techniques in Section 7.2.1

Acquisition Criterion Boyle uses the Maximum Expected Improvement (MEI) acquisition criterion given in Equation 2.33 with $\xi = 0$ throughout his work. He also proposes a "local" version of MEI, where the next point to acquire is constrained to be within distance ϵ of the last point acquired. He proposes a schedule for ϵ using an increment $\epsilon_{\text{inc}} > 1$ as follows:

$$\epsilon_{i+1} = \begin{cases} \epsilon_i \times \epsilon_{\text{inc}} & \text{if the last acquisition was an improvement} \\ \epsilon_i / \epsilon_{\text{inc}} & \text{otherwise} \end{cases} \tag{2.44}$$

Details of setting ϵ_0 and ϵ_{inc} are left to the user.

Empirical Results The various modifications by Boyle are tested on several analytically constructed problems, each of which consists of a single d-dimensional, possibly non-axis-aligned Gaussian function. Dimensions of the various test cases range from 1 to 36. Note that each of these functions has a single local optimum that is also the global optimum.

Boyle found improved performance when using a covariance function that is "matched" to the objective. That is, using an axis-aligned model with an axis-aligned objective or

using a non-axis-aligned model with a non-axis-aligned objective gave the best performance. Interestingly, he also found that results when the two models were averaged according to their plausibility were better than either used alone. This model-averaging approach was also applied to an 8-dimensional "double pole balancing" control problem, for which Gaussian process based optimization found good controllers using fewer evaluations than a competing genetic algorithm technique.

2.2 Gaussian Processes in Machine Learning

The preceding methods were developed over a number of years by researchers in the statistical sciences familiar with general random field models, and by researchers in the engineering sciences familiar with the "kriging" models of geostatistics. Gaussian processes have become widely popular in the machine learning community only during the last decade. Advances have been made adapting the GP model to various tasks in regression, classification, and reinforcement learning. In the process of these adaptations, various insights and tricks for the practical implementation and use of GPs have been developed. Some techniques particularly relevant to Bayesian optimization with GPs are geared toward improving model selection, and reducing the computation time necessary for inference.

The specification of a Gaussian Process prior involves choosing a prior mean function $\mu_0(x)$ and a covariance function or kernel $k(x, x)$. We have seen two primary examples of possible kernels: Mockus's Wiener process kernel

$$k(x, z) = \sigma_f^2 \prod_{i=1}^{d} \left(1 - \frac{x_i - z_i}{2}\right) \tag{2.45}$$

and the "squared exponential" kernel used in EGO

$$k(x, z) = \sigma_f^2 \cdot \mathrm{e}^{-\sum_{i=1}^{d} \theta_i |x_i - z_i|^{p_i}} \tag{2.46}$$

These two kernels give rise to very different sampled functions. When $\forall i \;\; p_i = 2$ in kernel (2.46), for example, functions sampled from the resulting Gaussian process prior are infinitely differentiable. Functions sampled from a Gaussian process prior using kernel (2.45) on the other hand are nowhere differentiable. Even using only kernel (2.46), we can produce a wide variety of priors by adjusting the free parameters p and θ. If $1 \leq p < 2$, the sampled functions become non-differentiable again, and are progressively "rougher" as we decrease p. On the other hand, θ_i controls the "length scale" of the process. This is a larger-scale

property that can be related to the distribution of the number of zero-crossings of the function on a given interval. As we increase θ_i, sampled functions become increasingly smooth along the ith axis, in the sense that the expected number of zero crossings decreases [36].

The most common version of (2.46) is a slightly modified case where $\forall i \;\; p_i = 2$.

$$k(x, z) = \sigma_f^2 \cdot \mathrm{e}^{-\frac{||r||^2}{2}} \tag{2.47}$$

where r is the vector given by

$$r_i(x_i, z_i) = \frac{x_i - z_i}{\ell_i}, \; i = 1 \ldots d$$

This is known as the Gaussian kernel, the ARD kernel, or the *squared exponential* kernel, which is how we will refer to it.

Another commonly used kernel is the Matérn kernel, which consists of an exponential kernel multiplied by a polynomial. These are parameterized by an additional real parameter $\nu > 0$; however, only two values of ν are commonly used in machine learning applications

$$k_{\nu=3/2}(r) = \sigma_f^2 \cdot \left(1 + \sqrt{3}||r||\right) \cdot \mathrm{e}^{-\sqrt{3}||r||} \tag{2.48}$$

$$k_{\nu=5/2}(r) = \sigma_f^2 \cdot \left(1 + \sqrt{5}||r|| + \frac{5}{3}||r||^2\right) \cdot \mathrm{e}^{-\sqrt{5}||r||} \tag{2.49}$$

where again, $r_i(x_i, z_i) = \frac{x_i - z_i}{\ell_i}$, $i = 1 \ldots d$. These kernels produce processes that are mean-square differentiable exactly $\lfloor \nu \rfloor$ times, and as $\nu \to \infty$ the Matérn kernel tends toward the squared exponential kernel described earlier [36].

Because of the degree of flexibility and lack of *a priori* knowledge about how to select a kernel or its parameters, it is common in the machine learning community to use a maximum likelihood criterion to choose these. The Gaussian likelihood function is given by

$$\log p(F_{\mathbf{x}} = \mathbf{f}) = -\frac{1}{2}\mathbf{f}^{\mathsf{T}} \, k(\mathbf{x}, \mathbf{x})^{-1} \, \mathbf{f} - \frac{1}{2}\log |k(\mathbf{x}, \mathbf{x})| - \frac{n}{2}\log 2\pi \tag{2.50}$$

Empirical results have shown that using the maximum likelihood criterion for choosing a kernel works well in many cases. A popular first attempt at modeling is to use the squared exponential kernel and adapt the ℓ_i parameters using maximum likelihood. This is known as the "Automatic Relevance Detection" or ARD approach [28]. Success with ARD has resulted in still more ambitious parameterizations of kernels. For example, kernel (2.47) can be extended as follows:

$$k(x, z) = \sigma_f^2 \cdot \mathrm{e}^{-(x-z)^{\mathsf{T}}\Theta(x-z)} \tag{2.51}$$

Here, Θ is a $d \times d$ positive semi-definite matrix. If Θ is diagonal, (2.51) is equivalent to (2.47) with all p_i set to 2. Estimating the full matrix Θ is in many cases infeasible if data are limited; however, if we assume Θ decomposes as

$$\Theta = \Lambda\Lambda^{\mathsf{T}} + \text{diag}(\boldsymbol{\theta})^{-2} \tag{2.52}$$

where $\boldsymbol{\theta}$ is a vector of positive values and Λ is $d \times m$, $m < d$, then we have significantly fewer parameters than if we were trying to estimate a more general Θ, but we can still identify interactions between input dimensions. This is called the *factor analysis* approach [36], and allows us to describe length-scales of the function in a limited number (m) of non-axis-aligned directions. The kernel is therefore similar to but not expressive as that of Boyle described in Section 2.1.5. This approach has been used effectively to provide more flexibility in modeling than the axis-aligned approach (i.e. kernel (2.47)) while controlling the number of parameters that must be fit.

2.3 Extrema of Gaussian Processes

The behaviour of Gaussian[8] processes has been studied extensively over the past 100 years or so, in a way that combines questions about the behaviour of functions with questions about the behaviour of random variables.

Common questions attempt to somehow encapsulate information about the behaviour of the function in a single number. For example, how many zeroes does the function in question have? How many critical points does it have? What is its global maximum? For random processes, these questions are of course ideally answered with a distribution instead of a single value; unfortunately, in many cases we are only able to give an analytic answer for the first moment of the distribution or for an approximation to its tail probabilities. Furthermore, investigating these quantities soon reveals that a treatment of general Gaussian processes is impractical; therefore before we begin we introduce three definitions to restrict our attention to *stationary* processes, *isotropic* processes, and *axis-scaled isotropic* processes.

Definition 1 (Stationarity). A real-valued random field $F(x)$, $x \in \mathbb{R}^d$ is *stationary* [1] (or *homogeneous*) if its finite dimensional distributions are invariant under translations of the parameter x. That is, for any set of points $x^1, x^2, ..., x^k$ and any point z (all in $\mathbb{R}^d$), the

[8] Many of the results we review here apply to some degree to general random processes; however we restrict our discussion to the Gaussian case.

distribution of

$$[F_{x^1}, F_{x^2}, ..., F_{x^k}]$$

is identical to that of

$$[F_{x^1+z}, F_{x^2+z}, ..., F_{x^k+z}]$$

Two immediate consequences of this requirement are that for all x and z,

$$\mathrm{E}[F(x)] \equiv c \text{ where } c \text{ is some constant}$$

$$\mathrm{E}[(F_x - c)(F_z - c)] \text{ is a function of } x - z \text{ only}$$

For real-valued processes, these two properties taken together imply stationarity. (See Theorem 2.1.1 by Adler [1].) Note that we also apply the adjective *stationary* to kernels that are a function of $x - z$ only; in this case we may write the kernel as a function of one variable, $k(r)$, where $r = x - z$.

All of the *prior* Gaussian processes we deal with in this work will be stationary, though posterior processes will not be.

Definition 2 (Isotropy). A real-valued random field $F(x)$, $x \in \mathbb{R}^d$ is *isotropic* if it is stationary and

$$\mathrm{E}[(F_x - c)(F_z - c)] \text{ is a function of } ||x - z|| \text{ only}$$

where $||\cdot||$ is the Euclidean 2-norm [1].

Finally, for convenience we define a slightly broader class of processes.

Definition 3 (Axis-Scaled Isotropy). A real-valued random field $F(x)$, $x \in \mathbb{R}^d$ is *axis-scaled isotropic* if it is stationary and

$$\mathrm{E}[(F_x - c)(F_z - c)] \text{ is a function of } ||(x - z)\,\mathrm{L}^{-1}|| \text{ only}$$

where L is a $d \times d$ matrix with $\ell_{ij} = 0$ where $i \neq j$ and $\ell_{ii} > 0$ are characteristic length-scales for each dimension (usually written ℓ_i.)

It is convenient to define the kernel of an axis-aligned isotropic process as a function $k(r)$ where

$$r_i(x_i, z_i) = \frac{x_i - z_i}{\ell_i}, \; i = 1 \ldots d$$

but to have the function only depend on r through its norm, $||r||$. This last class of processes corresponds to those used in the Automatic Relevance Determination procedure described above.

2.3.1 Spectral representation

Here, we give a *very* brief overview of the ideas behind spectral representations for Gaussian processes. More complete treatments can be found in many sources [1, 21, 36, 2]. The spectral theory of Gaussian processes plays a critical role in the understanding of many of their properties, because it allows us to decompose them into sinusoidal components for which those same properties (related to amplitude, periodicity, etc.) are more easily accessible. This is particularly helpful when there is one dominant sinusoidal component, as there typically is in our models, since expectations of quantities like level crossings and local maxima (described later) depend only on this easily computed component.

All of the results that follow are based in part on the fact that any real-valued stationary Gaussian process with zero mean can be expressed as follows:

$$F_x = \int_{\mathbb{R}^d} [\cos(\lambda \cdot x)\ \mathrm{d}U(\lambda) - \sin(\lambda \cdot x)\ \mathrm{d}V(\lambda)] \tag{2.53}$$

Note that here, U and V are random fields that take intervals (i.e. rectangles) of $\mathbb{R}^d$ as inputs, making this a stochastic integral. A complete explanation of the semantics of this equation is given by Adler [1], but here we just want to give an intuition of the consequence of this equivalence. Note that we can approximate (2.53) by a sum of sinusoidal components

$$F_x \approx \sum_i [\cos(\lambda^{(i)} \cdot x) U(\Lambda^{(i)}) - \sin(\lambda^{(i)} \cdot x)\ V(\Lambda^{(i)})] \tag{2.54}$$

Here, each $\Lambda^{(i)}$ is an interval containing the point $\lambda^{(i)}$, $\Lambda^{(i)} \cap \Lambda^{(j)} = \emptyset$ for $i \neq j$, and $\cup_i \Lambda^{(i)} = \mathbb{R}^d$. We can see that the field F can be approximately expressed as a sum of sinusoidal components, each one with amplitude $\sqrt{U(\Lambda^{(i)})^2 + V(\Lambda^{(i)})^2}$ and frequency determined by $\lambda^{(i)}$.

The *spectral distribution* of a Gaussian process is defined as

$$S(\lambda) = \mathrm{E}[|U(\lambda) + \imath V(\lambda)|^2] \tag{2.55}$$

which is the expected squared magnitude at any given frequency λ. (Here, $\imath^2 = -1$. Also note that S is not necessarily a probability measure.) The second-order[9] *spectral moments* are given by

$$\lambda_{ij} = \int_{\mathbb{R}^d} \lambda_i \lambda_j\ \mathrm{d}S(\lambda) = -\left.\frac{\partial^2 k(r)}{\partial r_i \partial r_j}\right|_{r=0} \tag{2.56}$$

Here, $k(r)$ is the kernel of the process. The quantities λ_{ii} indicate the squared frequencies along each axis that have the highest expected amplitude, scaled by $k(0) = \int \mathrm{d}S(\lambda)$. This is

[9] Note that the first-order spectral moments are identically zero [1].

useful particularly in the isotropic case where $\lambda_{ij:i\neq j} = 0$. The right hand side of the above expression is true because the spectral distribution also completely determines the kernel in the following way

$$k(r) = \int_{\mathbb{R}^d} \cos(r \cdot \lambda) \, \mathrm{d}S(\lambda)$$

For this work, we will never deal with S, U, or V directly; however, the spectral representation provides us with two critical pieces of information: A stationary Gaussian process can be thought of as a random sum of sinusoids, and the most important frequency of these sinusoids can be recovered simply by differentiating the kernel function. This is important because, as we will see, many expectations of (quasi-)periodic events (level crossings, local maxima, etc.) depend only on the "dominant" frequency of the function in question.

2.3.2 Stationary processes in one dimension

The first extensively studied Gaussian processes were those indexed by $\mathbb{Z}$ or $\mathbb{R}$, usually referred to as discrete or continuous time series, respectively. The study of time series was one of the first forays into the analysis of sets of random variables that are not independent and identically distributed—their dependency structure is explicitly stated. In the Gaussian case, this structure is defined by a kernel as we saw earlier

Considering the one-dimensional case has the advantage that many interesting quantities have easily computable, analytic answers that give further insight into how the parameters of a process influence different properties of that process.

Level Crossings One useful measure of the complexity of a time series is the number of *u-level crossings* of sampled functions. A u-level crossing is a point on a function f where $f(x) = u$ and $f'(x) \neq 0$. This quantity is used as a measure of the oscillatory properties of a function or process, and is related to the "dominant frequency" of the object in question [21].

Theorem 1 (The Rice Formula). *Let F be a zero-mean, unit-variance Gaussian process on the interval $\mathcal{I} \subset \mathbb{R}$ with kernel $k(r)$. Then the expected number of u-level crossings of F over $\mathcal{I}$, denoted $N_u^{\mathcal{I}}$, is given by*

$$\mathrm{E}[N_u^{\mathcal{I}}] = \frac{\mu_L(\mathcal{I})}{\pi} \sqrt{\frac{-\left.\frac{\partial^2 k(r)}{\partial r^2}\right|_{r=0}}{k(0)}} e^{-u^2/k(0)} \tag{2.57}$$

provided the second derivative of k exists at 0[1].

This formula is perhaps the most famous in time series analysis, and has been proven many times and in many ways [1, 21, 36, 2]. We omit the proof here, but provide an analogous proof for the expected number of local maxima in the next section. It is worth noticing that the quantity $\omega = \sqrt{-k''(0)/k(0)}$ is the normalized second spectral moment, which as we mentioned earlier can be thought of as the dominant frequency of the Gaussian process. Therefore, if we take $u = 0$ and $\mathcal{I} = [0, 1]$, say, then the expected number of 0-level crossings is ω/π. For comparison, the number of 0-level crossings of a sinusoid with frequency ω is $\in \{\lfloor \omega/\pi \rfloor, \lfloor \omega/\pi \rfloor + 1\}$.

Higher-Order Crossings We now consider what are known as "higher-order crossings" [21]. The "order" referred to here is the order of the derivative of the process we are considering. The study of higher-order crossings brings up an important property: If a Gaussian process is c times mean square differentiable, then each of its partial derivatives up to order c is also a Gaussian process. Furthermore, the original process and its derivatives are all jointly Gaussian, with covariances given by

$$\operatorname{Cov}\left(\frac{\partial^\alpha F(x)}{\partial x_{i_1}^{j_1}\partial x_{i_2}^{j_2}\dots\partial x_{i_m}^{j_m}}, \frac{\partial^\beta F(\mathbf{z})}{\partial z_{k_1}^{l_1}\partial z_{k_2}^{l_2}\dots\partial z_{k_n}^{l_n}}\right) = \frac{\partial^{\alpha+\beta}k(\mathbf{x},\mathbf{z})}{\partial x_{i_1}^{j_1}\partial x_{i_2}^{j_2}\dots\partial x_{i_m}^{j_m}\partial z_{k_1}^{l_1}\partial z_{k_2}^{l_2}\dots\partial z_{k_n}^{l_n}} \tag{2.58}$$

where $\alpha = \sum_\gamma i_\gamma$ and $\beta = \sum_\delta l_\delta$, $\partial^0 f(x)/\partial x = f(x)$, $\alpha \le c$ and $\beta \le c$. Briefly, the justification for this is an interchanging of expectation and differentiation, both of which are linear operators. Again, more thorough proofs can be found in the literature [2].

Theorem 2 (Expected Number of Local Maxima). *The expected number of local maxima (denoted $N^\frown(\mathcal{I})$) of a twice mean square differentiable, stationary Gaussian process over an interval I is given by*

$$\mathrm{E}[N^\frown(\mathcal{I})] = \frac{\mu_L(\mathcal{I})}{2\pi}\sqrt{\frac{\left.\frac{\partial^4 k(r)}{\partial r^4}\right|_{r=0}}{\left.-\frac{\partial^2 k(r)}{\partial r^2}\right|_{r=0}}} \tag{2.59}$$

where $\mu_L(\mathcal{I})$ is the length (Lebesgue measure) of the domain of x, and k is the kernel, which depends only on the distance $r = x - z$ between domain points.

Proof. We could simply use Rice's formula 2.57, if we knew that

1. Local maxima are 0-level downcrossings of the first derivative
2. There are half as many u-level downcrossings as u-level crossings
3. The covariance of the first derivative process is given by (2.58)

However, we present a proof that gives a feel for how the expectations of numbers of level crossings and other similar events are computed for Gaussian processes. We will closely follow the approach taken by Adler and Taylor [2] by computing the number of downcrossings of a deterministic, one-dimensional function and then taking expectations to recover this version of Rice's formula. This approach affords a straightforward, intuitive proof, but relies on heavy use of Dirac delta functions and changing orders of integration in ways that are not immediately clearly justifiable. We refer the intrepid reader who is interested in the detailed justification of these operations to investigate the aforementioned work.

We define a "downcrossing" $x^{\downarrow}$ of a one-dimensional deterministic function g to be a point where $g(x^{\downarrow}) = 0$ and $g'(x^{\downarrow}) < 0$. We presume that these points are isolated: For any downcrossing $x_{\downarrow}$, there is an interval $\mathcal{I}$ inside which $x_{\downarrow} \in \mathcal{I}$ is the only downcrossing and $\forall x \in \mathcal{I}, g'(x) < 0$.

Let δ be the Dirac delta function which has the property

$$\int_{\mathbb{R}} \delta(x) f(x) \mathrm{d}x = f(x)$$

for "reasonable" functions f. Then within the interval $[z, w]$ that contains a single downcrossing, we have

$$\int_z^w -\delta(f(x)) \cdot f'(x)\ \mathrm{d}x = \int_{f(z)}^{f(w)} -\delta(y)\ \mathrm{d}y = 1$$

by substituting $y = f(x)$ and $\mathrm{d}y = f'(x)\ \mathrm{d}x$. The negative sign appears because by assumption f is decreasing over $\mathcal{I}$ so $f(z) < f(w)$. By integrating over the domain of f and using an indicator function to identify areas where $f'(x) < 0$, we can count the number of downcrossings $N^{\downarrow}$ as follows:

$$N^{\downarrow}(\mathcal{I}) = \int_{\mathcal{I}} -\delta(f(x)) \cdot \mathbf{1}_{(-\infty,0)}[f'(x)] \cdot f'(x)\ \mathrm{d}x$$

Here, $\mathbf{1}_{(-\infty,0)}[\cdot]$ is the indicator function for negative numbers. An immediate consequence of this is that we can count the number of interior local maxima of f, since these are simply points where $f'(x) = 0$ and $f''(x) < 0$. Hence, we have

$$N^{\frown}(\mathcal{I}) = \int_{\mathcal{I}} -\delta(f'(x)) \cdot \mathbf{1}_{(-\infty,0)}[f''(x)] \cdot f''(x)\ \mathrm{d}x$$

We now substitute the random process field $F(x)$ for the function $f(x)$, and take the expec-

tation

$$\begin{aligned}
\mathrm{E}[N^{\frown}(\mathcal{I})] &= \mathrm{E}\left[\int_{\mathcal{I}} -\delta(F'(x)) \cdot \mathbf{1}_{(-\infty,0)}[F''(x)] \cdot F''(x)\, \mathrm{d}x\right] \\
&= \int_{-\infty}^{0}\int_{-\infty}^{\infty}\int_{\mathcal{I}} -z\,\delta(y)p(F'(x)=y, F''(x)=z)\, \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z \\
&= \int_{-\infty}^{0}\int_{\mathcal{I}} -z\,p(F'(x)=0, F''(x)=z)\, \mathrm{d}x\, \mathrm{d}z \qquad (2.60)
\end{aligned}$$

Here, p is the joint probability density of $F'(x) = y, F''(x) = z$. Recall that in our case, this density is Gaussian, stationary (i.e. it does not depend on x) and y and z are independent. Therefore we can simplify and get

$$\begin{aligned}
\mathrm{E}[N^{\frown}(\mathcal{I})] &= \int_{\mathcal{I}} \mathrm{d}x \int_{-\infty}^{0} -z\,p(F'(x)=0, F''(x)=z)\, \mathrm{d}z \\
&= \mu_L(\mathcal{I}) \int_{-\infty}^{0} -z\,p(F'(x)=0)\cdot(F''(x)=z)\, \mathrm{d}z \\
&= \mu_L(\mathcal{I}) \cdot p(F'(x)=0)\cdot\int_{-\infty}^{0} -z\cdot(F''(x)=z)\, \mathrm{d}z \\
&= \mu_L(\mathcal{I}) \cdot \frac{1}{\sqrt{2\pi\sigma^2(F'(x))}}\cdot\int_{-\infty}^{0} -z\frac{e^{-z^2/2\sigma^2(F''(x))}}{\sqrt{2\pi\sigma^2(F''(x))}}\, \mathrm{d}z \\
&= \mu_L(\mathcal{I}) \cdot \frac{1}{\sqrt{2\pi\sigma^2(F'(x))}}\cdot\sqrt{\frac{\sigma^2(F''(x))}{2\pi}} \\
&= \mu_L(\mathcal{I}) \cdot \frac{1}{2\pi}\sqrt{\frac{\sigma^2(F''(x))}{\sigma^2(F'(x))}}
\end{aligned}$$

We can then use a simplified version of (2.58) to determine that the marginal variance of the derivatives are given by $\sigma^2(F^{(i)}(x)) = (-1)^i \cdot k^{(2i)}(0)$. Therefore,

$$\mathrm{E}[N^{\frown}(\mathcal{I})] = \mu_L(\mathcal{I}) \cdot \frac{1}{2\pi}\sqrt{\frac{\left.\frac{\partial^4 k(r)}{\partial r^4}\right|_{r=0}}{-\left.\frac{\partial^2 k(r)}{\partial r^2}\right|_{r=0}}} \qquad (2.61)$$

This gives the exact expected number of local maxima of functions drawn from a sufficiently smooth stationary Gaussian process over a closed interval $I \subset \mathbb{R}$. □

2.3.3 Stationary processes in many dimensions

We now examine processes over subsets $\mathcal{I} \subset \mathbb{R}^d$. First- and second-order mean square partial derivatives of F are denoted $F_i(x) = \partial F(x)/\partial x_i$ and $F_{ij}(x) = \partial^2 F(x)/\partial x_i \partial x_j$, respectively.

Expected Number of Local Maxima In principle, we can count the number of local maxima of a function in d dimensions by computing

$$N^{\frown}(\mathcal{I}) = \int_{\mathcal{I}} \delta(\nabla_f(x)) \cdot \mathbf{1}_{\prec 0}[\nabla_f^2(x)] \cdot |\det \nabla_f^2(x)|\, \mathrm{d}x$$

where $\mathbf{1}_{\prec 0}[\cdot]$ is the indicator function for negative definite matrices, $\delta(x) = \prod_i \delta(x_i)$, and $\nabla_f(x)$ and $\nabla_f^2(x)$ are the gradient and Hessian of f, respectively. This follows again from a variable substitution of the now multi-dimensional delta function, recalling that $\nabla_f^2(x)$ is the Jacobian of $\nabla f(x)$. Taking the expectation of this quantity over F, we can make similar simplifications as in the one-dimensional case, up to a point, and get:

$$\begin{aligned} \mathrm{E}[N^{\frown}(\mathcal{I})] &= \mu_L(\mathcal{I}) \int_{\mathrm{H} \prec 0} p(\nabla_F(x) = \mathbf{0}, \nabla_F^2(x) = \mathrm{H}) \cdot |\det \mathrm{H}| \, d\mathrm{H} \qquad (2.62) \\ &= \mu_L(\mathcal{I}) \cdot \prod_{i=1}^{d} p(F_i(x) = 0) \int_{\mathrm{H} \prec 0} p(\nabla_F^2(x) = \mathrm{H}) \cdot |\det \mathrm{H}| \, d\mathrm{H} \qquad (2.63) \end{aligned}$$

There are two extremely unfortunate characteristics of equation (2.63) that bear mentioning. First, the integral in question is over all possible negative definite matrices, which is a difficult region to express analytically in terms of the elements of H. Second, the elements of $\nabla_F^2(x)$ are dependent, so we need to deal with the full joint Gaussian distribution over Hessians, which has $d(d+1)/2$ variables. Even computing a simple orthant probability (i.e. $p(x > 0)$) has no known closed form for more than 4 dimensions, so it seems unlikely that a simple closed form for (2.63) exists.

In a similar vein, Adler [2] defines the quantities

$$\mu_k = \#\{x \in \mathcal{I} : f(x) \geq u, \ \nabla_f(x) = 0, \ \mathrm{index}(\nabla_f^2(x)) = k\} \qquad (2.64)$$

where the *index* of a matrix is the number of negative eigenvalues of that matrix. Clearly, if we set $u = -\infty$ then $\mu_d = N^{\frown}(\mathcal{I})$. Adler states categorically that "...it is actually *impossible* to obtain closed expressions for any of the $\mathrm{E}[\mu_k]$." Although μ_d is a slightly more general quantity than $N^{\frown}(\mathcal{I})$ since it involves the additional constraint $F(x) \geq u$, it seems that the prospects of obtaining an analytic answer for $N^{\frown}(\mathcal{I})$ are slim to none.

It is, however possible to approximate the number of local maxima that exceed a value u, which we denote $N_u^{\frown}$. An asymptotic expression for this quantity was given by Adler [1].

$$\mathrm{E}[N_u^{\frown}] = e^{-u^2/2\sigma_f^2} \frac{\mu_L(J)|\Lambda_J|^{1/2}}{(2\pi)^{(d+1)/2}\sigma_f^d} \left(\frac{u}{\sigma_f}\right)^{d-1} \left[1 + \mathcal{O}\left(\frac{1}{u/\sigma_f}\right)\right] \qquad (2.65)$$

It is important to note that this will count the number of *interior* local maxima, i.e. points with a zero gradient and negative definite Hessian. In the realm of constrained optimization, these do not encompass all of the maxima that are of interest—maxima that are against active constraints do not have this property, in general. We shall see how a different approach based on geometry rather than calculus can lead to quantities more appropriate for our use.

2.3.4 The Euler characteristic of excursion sets

Given the limitations of the critical-point, calculus-based approach to excursion theory, it is fortunate that there are other properties of Gaussian processes over high dimensional spaces that give insight into the behaviour of the "bumpiness" of F, have closed forms, and are at least somewhat tractable. The most prominent approach in the field has been to shift focus away from finding critical points (i.e. points where $\nabla_f(x) = 0$) and toward computing properties of *excursion sets*, which are sets $\mathcal{A}_u(f) = \{x : f(x) \geq u\}$.

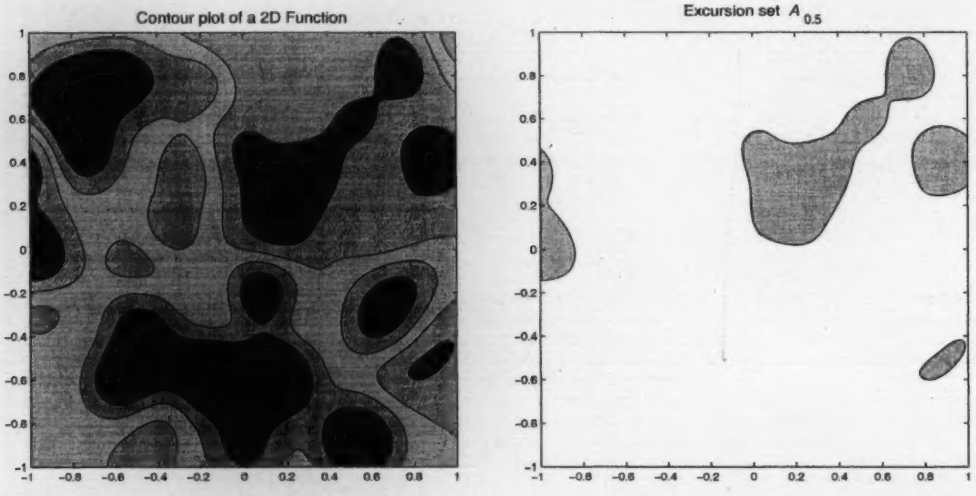


Figure 2.1: Example of an excursion set for a two-dimensional function f. Shown on the right in green, the excursion set $\mathcal{A}_{0.5}$—the set of domain points where $f > 0.5$—has four components, none of which have holes. Therefore $\chi(\mathcal{A}_{0.5}) = 4$ for this function.

Some examples of excursion sets in two dimensions are shown in Figure 2.1. For a function f with a "mountain range" type of behaviour, like samples from a smooth Gaussian process or a sinusoid, the excursion set $\mathcal{A}_u$ for $u = \sup_{\mathcal{I}} f - \delta$ for some small enough $\delta > 0$ will consist of a connected, nearly ellipsoid-shaped closed set of points containing $x^* = \text{argsup}_{\mathcal{I}} f(x)$, if we suppose that x^* is unique and is not on the boundary of $\mathcal{I}$. To see this, note that by Taylor's theorem, the behaviour of f about the point x^* will be very nearly quadratic within a small enough radius, and that the level sets of a quadratic function are ellipsoids. As we decrease u, the size of $\mathcal{A}_u$ will grow, and new components will appear around other local maxima that u crosses on its descent. $\mathcal{A}_u$ will become more and more connected until it consists of most of the domain $\mathcal{I}$, but with holes surrounding the locations of extreme local minima.

We now consider what we can tell about a function from looking at its excursion sets to

try to give an idea of their potential in studying extrema. First, it is easy to see that the number of connected components of $\mathcal{A}_u$ gives a lower bound on the number of local maxima that exceed level u. Second, if we somehow knew that a Gaussian process would produce either a single local maximum above u or none at all, then the number of components of $\mathcal{A}_u$ for any sampled function will be zero or one, and the expected number of components of $\mathcal{A}_u$ would be equal to $p(\sup_x F_x \geq u)$, which we call the *excursion probability* of F above u. It turns out that the best asymptotic approximations known for excursion probabilities (and they are very good for moderate to large u/σ_f, i.e. $u/\sigma_f > 1.5$) are related to counting connected components of excursion sets.

The number of connected components of a closed set is an example of a topological invariant. That is, the quantity is the same for all sets that are *homeomorphic*—sets between which there exists a continuous, one-to-one, and onto mapping. Less formally, a topological invariant property of two sets will be the same if one set can be smoothly deformed into the other, like a doughnut and a coffee cup, but not a doughnut and a bowling ball[10]. The *Euler characteristic* χ is perhaps the most famous topological invariant[11] Precisely defining the Euler characteristic is a somewhat lengthy process, so we instead give a few of its properties that are relevant to our discussion.

$$\begin{aligned} \chi(\mathcal{B}_k) &= 1 \text{ where } \mathcal{B}_k \text{ is the unit ball in } \mathbb{R}^k \\ \chi(\mathcal{X} \cup \mathcal{Y}) &= \chi(\mathcal{X}) + \chi(\mathcal{Y}) \text{ if } \mathcal{X} \cap \mathcal{Y} = \emptyset \end{aligned}$$

So, any set that is homeomorphic to $\mathcal{B}_k$, for example the ellipsoid-shaped components of excursion sets around local maxima that we mentioned above, counts for 1 when computing the Euler characteristic. Furthermore, the characteristic is additive, so if $\mathcal{A}_u$ consists of some number of disjoint components that are each homeomorphic to $\mathcal{B}_k$, then the $\chi(\mathcal{A}_u)$ will be the number of components of $\mathcal{A}_u$. We have of course left out all of the sets which are not homeomorphic to $\mathcal{B}_k$; here are a few examples of these:

$$\begin{aligned} \chi(\mathcal{K}_{k,h}) &= 1 + h \cdot (-1)^N \\ \chi(\bar{\mathcal{K}}_{k,h}) &= 1 - h \end{aligned}$$

Here, $\mathcal{K}_{k,h}$ is $\mathcal{B}_k$ with h non-intersecting k-dimensional holes drilled through it, and $\bar{\mathcal{K}}_{k,h}$ is

[10]The problem here is continuity—to map the basketball back to the doughnut, there will have to be points that are nearby on the bowling ball that map to points that are far apart on the original doughnut, and vice versa. This is caused by the hole in the doughnut. (Or by the absence of a hole through the bowling ball.)

[11]Along with *genus*, which is defined for surfaces. The orientable genus g can be defined as $\chi = 2 - 2g$.

$\mathcal{B}_k$ with h "handles" attached to it. Envisioning a k-dimensional sphere with k-dimensional handles attached to it is taxing at best, but the important thing to note is that $\chi(\mathcal{A})$ is *not* simply the number of connected components of $\mathcal{A}$.

We now come to the single most important formula governing the extrema of Gaussian processes.

Theorem 3. *For a stationary Gaussian process with zero mean, we have*

$$\mathrm{E}[\chi(\mathcal{A}_u)] = e^{-u^2/2\sigma_f^2} \sum_{k=1}^{d} \sum_{\mathcal{J}\in\mathcal{E}_k} \frac{\mu_L(\mathcal{J})|\Lambda_{\mathcal{J}}|^{1/2}}{(2\pi)^{(k+1)/2}\sigma_f^k} H_{k-1}\left(\frac{u}{\sigma_f}\right) + \Psi\left(\frac{u}{\sigma_f}\right) \tag{2.66}$$

where the process is over the set $\mathcal{I}$, a rectangle in $\mathbb{R}^d$ with one vertex at the origin. The $\mathcal{E}_k$ are collections of the k-dimensional facets of the boundary of $\mathcal{I}$ that contain the origin, of which there are $\binom{d}{k}$. For example, if $\mathcal{I} \subset \mathbb{R}^3$, then $\mathcal{E}_1$ would be the three line segments that border $\mathcal{I}$ and touch the origin, $\mathcal{E}_2$ would be the three 2-dimensional rectangles that border $\mathcal{I}$ and touch the origin, and $\mathcal{E}_3$ would be $\mathcal{I}$ itself. $\Lambda_{\mathcal{J}}$ is the matrix of spectral moments of the process when restricted to the set $\mathcal{J}$, and σ_f is the process variance. As before, $\mu_L(\mathcal{J})$ is the (k-dimensional) Lebesgue measure of the set $\mathcal{J}$, and $\Psi(x)$ is the standard normal right tail probability $p(X > x)$, $X \sim \mathcal{N}(0,1)$. $H_{k-1}(\cdot)$ is the $k-1$st "probabilist's" Hermite polynomial.[12]

The preceding formula, given by Adler [2], is important not merely because it is interesting to know the expected Euler characteristic of excursion sets, but because it provides the state-of-the-art approximation to excursion probabilities of Gaussian processes:

$$|p\{\sup F(x) \geq u\} - \mathrm{E}[\chi(\mathcal{A}_u)]| < \mathcal{O}\left(e^{-\alpha u^2/2\sigma_f^2}\right) \tag{2.67}$$

where $\alpha > 1$. Furthermore, it provides insight in to how the spectral moments—and therefore the length-scales—of a process influence its supremum distribution. To see this more clearly, we give a more specialised version of Equation (2.66).

Corollary 1. *For an axis-scaled isotropic Gaussian process over the hyper-rectangle $\mathcal{I} = [0, w_1] \times [0, w_2] \times \ldots \times [0, w_d]$, if we let $\mathcal{D}(\mathcal{J})$ represent the set of indices of the k axes that are included in a boundary set $\mathcal{J} \in \mathcal{E}_k$, the expected Euler characteristic is given by*

$$\mathrm{E}[\chi(\mathcal{A}_u)] = e^{-u^2/2\sigma_f^2} \sum_{k=1}^{d} \sum_{\mathcal{J}\in\mathcal{E}_k} \frac{[\prod_{i\in\mathcal{D}(\mathcal{J})} w_i] \cdot [\prod_{i\in\mathcal{D}(\mathcal{J})} \sqrt{\lambda_{ii}}]}{(2\pi)^{(k+1)/2}\sigma_f^k} H_{k-1}\left(\frac{u}{\sigma_f}\right) + \Psi\left(\frac{u}{\sigma_f}\right) \tag{2.68}$$

[12] See the List of Symbols for some properties of H.

Proof. This corollary uses two main properties: First, for an axis-scaled isotropic process, since $\lambda_{ij:i\neq j} = 0$, we have $|\Lambda_{\mathcal{J}}|^{1/2} = \prod_{i\in\mathcal{D}(\mathcal{J})}\sqrt{\lambda_{ii}}$. Furthermore, since the domain is a hyper-rectangle, the k-dimensional Lebesgue measure of each of these is given by $\mu_L(\mathcal{J}) = \prod_{i\in\mathcal{D}(\mathcal{J})} w_i$. □

Recall from (2.66) that there are $\binom{d}{k}$ such boundary sets $\mathcal{J}$ for each $k \in \{1, 2, ..., d\}$. Therefore, a naïve expansion of the sum in (2.68) would involve $\sum_{i=1}^{d}\binom{d}{k} = 2^d$ terms, making use of this approximation prohibitively expensive. We now give a novel re-formulation of (2.66) that takes $\mathcal{O}(d^2)$ time.

Theorem 4. *For an axis-scaled isotropic Gaussian process as defined in Corollary 1, the expected Euler characteristic is also given by*

$$\mathrm{E}[\chi(\mathcal{A}_u)] = e^{-u^2/2\sigma_f^2}\sum_{k=1}^{d}\frac{S_k}{(2\pi)^{(k+1)/2}\sigma_f^k}H_{k-1}\left(\frac{u}{\sigma_f}\right) + \Psi\left(\frac{u}{\sigma_f}\right) \tag{2.69}$$

where

$$S_i = \frac{\sum_{k=1}^{i}(-1)^{k+1}S_{i-k}\cdot S^{[k]}}{i} \tag{2.70}$$

$$S_0 = 1 \tag{2.71}$$

$$S^{[a]} = \sum_{i=1}^{d}(w_i\sqrt{\lambda_{ii}})^a \tag{2.72}$$

This alternate factorization, in terms of S, generates the necessary sums for each dimension k in $\mathcal{O}(d)$ time, for a total run time of $\mathcal{O}(d^2)$. Each of σ_f^2, w_i, λ_{ii}, and H are defined as in Corollary 1.

Proof. First, note that the denominator inside the inner sum of (2.68) does not depend on $\mathcal{J}$, and so can immediately be factored out. This leaves us with the task of computing

$$\sum_{\mathcal{J}\in\mathcal{E}_k}\left(\left[\prod_{i\in\mathcal{D}(\mathcal{J})}\sqrt{\lambda_{ii}}\right]\cdot\left[\prod_{i\in\mathcal{D}(\mathcal{J})}w_i\right]\right) = \sum_{\mathcal{J}\in\mathcal{E}_k}\left(\prod_{i\in\mathcal{D}(\mathcal{J})}w_i\sqrt{\lambda_{ii}}\right) = \sum_{\mathcal{J}\in\mathcal{E}_k}\left(\prod_{i\in\mathcal{D}(\mathcal{J})}q_i\right)$$

for each k, where $q_i = w_i\sqrt{\lambda_{ii}}$. The sum is over all subsets of size k of the index set $\{1, 2, ..., d\}$, so for each k the sum has $\binom{d}{k}$ terms that cover all possible monomials of degree k formed from $\{q_1, q_2, ..., q_d\}$ that have *no exponent greater than 1.*

The sequence S_i constructs these sums as follows: at each step $k > 0$, all monomials of degree k composed of k distinct variables are generated, along with some that have a 2 in the exponent. These are then subtracted away in the next term, but this introduces some monomials with cubed variables which are removed by the term after and so on until we

remove the last of the extra terms using $S^{[i]}$. This can be be improved, given that S_d can be computed in linear time—it is simply $\prod_{i=1}^{d} w_i\sqrt{\lambda_{ii}}$. In practice, we also use the following sequence

$$\begin{aligned} S_i &= \frac{\sum_{k=i}^{d-1}(-1)^{k-i}S_{k+1}\cdot S^{[-(k-i+1)]}}{d-i} \\ S_d &= \prod_{i=1}^{d} w_i\sqrt{\lambda_{ii}} \\ S^{[-a]} &= \sum_{i=1}^{d}(w_i\sqrt{\lambda_{ii}})^{-a} \end{aligned}$$

which begins with S_d and successively removes variables from each monomial, thus introducing terms with extra negative powers, which are removed in a manner analogous to the previous sequence. We can save some time by using both sequences and "meeting in the middle"—we use the first sequence to compute $S_0 ... S_{d/2}$ and the second to compute $S_{d/2+1} ... S_d$. □

The values S_i are known as the ith *elementary symmetric polynomials*, evaluated at $(w_1\sqrt{\lambda_{11}}, w_2\sqrt{\lambda_{22}}, ..., w_d\sqrt{\lambda_{dd}})$. These are symmetric in all d variables, which implies that the ordering of the axes of the domain does not matter—so long as the $w_i\sqrt{\lambda_{ii}}$ remain the same, a process will have the same expected Euler characteristics. They are also linear in each w_i and $\sqrt{\lambda_{ii}}$, since there are no exponents greater than one. This means that if we fix u, σ_f, and w_i, then $\mathrm{E}[\chi(\mathcal{A}_u)] = c_1 \cdot \sqrt{\lambda_{ii}} + c_0$ for some constants c_1 and c_0.

For example, suppose we had a zero-mean Gaussian process with covariance given by the Matérn kernel with $\nu = 3/2$ as defined in Equation 2.48. To use Theorem 4, we need to know σ_f^2, which is part of the definition of the process, the w_i, which are the widths of each axis of the domain, and the λ_{ii} which are derived from the kernel function. Recall that for this Matérn kernel,

$$k_{\nu=3/2}(r) = \sigma_f^2 \cdot \left(1 + \sqrt{3}||r||\right) \cdot \mathrm{e}^{-\sqrt{3}||r||}$$

where

$$r_i(x_i, z_i) = \frac{x_i - z_i}{\ell_i},\ i = 1 \ldots d$$

Using Equation 2.56, we have

$$\begin{aligned} \lambda_{ii} &= -\left.\frac{\partial^2 k(r)}{\partial r_i^2}\right|_{r=0} \\ &= -\left.\frac{\partial^2}{\partial r_i^2}\left[\sigma_f^2 \cdot \left(1 + \sqrt{3}||r||\right) \cdot \mathrm{e}^{-\sqrt{3}||r||}\right]\right|_{r=0} \\ &= \cdot\ \sigma_f^2 \cdot \frac{3}{\ell_i^2} \end{aligned}$$

From this, we see that the spectral moments λ_{ii} depend only on the process variance and the length scales. (Incidentally, if we had used a squared exponential kernel for k instead, we would have $\lambda_{ii} = \sigma_f^2/\ell_i^2$.) This shows that decreasing the length scale along dimension i increases the spectral moment for that dimension, pushing more power into higher frequencies relative to the other dimensions, and making dimension i "bumpier" than the others.[13] Using σ_f, w_i, and λ_{ii} with Theorem 4, we can compute $E[\chi(\mathcal{A}_u)]$ for this kernel with arbitrary length scales and any value of u. Furthermore, since $E[\chi(\mathcal{A}_u)]$ is linear in each of the λ_{ii}, it is a smooth function with respect to ℓ_i and σ_f^2, and derivatives with respect to these quantities are easily computed using the chain rule. We will require these derivatives later when we use $E[\chi(\mathcal{A}_u)]$ in a prior over the ℓ_i that favours simple functions.

Theorem 4 allows the expected Euler characteristic to be used for the first time in a computational setting when working with Gaussian processes over hyperrectangles with more than a few dimensions. In the next chapter, we will detail a novel prior on process length scales that takes advantage of this capability.

Processes over more general spaces

The topologically inclined reader should note that many of the results presented here for processes over compact subsets of $\mathbb{R}^d$ have been extended to processes over more general sets and topologies, including locally convex, smooth manifolds. These extensions have allowed, for example, an analysis of the cosmic microwave background radiation which is frequently modeled as a Gaussian process over the sphere [29].

[13] Recall that the "dominant frequency" is given by the normalized second-order spectral moments, which are in this case λ_{ii}/σ_f^2.

Chapter 3

Practical Bayesian Optimization

During the development of any system, choices inevitably must be made that are unforeseen or glossed over in the initial abstract conception of that system. We will describe here in detail each choice that was made in the development of our global optimization system, the reasons for making it, and the consequences that follow from it. This includes the specific Gaussian process models used, our approach to parameter learning, the acquisition criteria we developed and investigated, and our approach to optimizing them.

3.1 Gaussian Process Model

The response model used throughout this work is essentially that described in Section 2.1.4; that is, we construct a full Gaussian process model using all of the currently available data points. The model uses a constant prior mean over the entire domain, and an axis-scaled isotropic kernel which may be either squared exponential or Matérn. We learn the length scale for each dimension using a likelihood function similar to the Automatic Relevance Determination (ARD) objective.

We allow for the inclusion of first-order derivative information as well. If gradient information about the target function is available, the gradient observations are used both in computing the posterior model and in computing the likelihoods used in parameter learning.

3.1.1 Parameter Learning

The models we consider have $d + 2$ parameters. Each of the dimensions $i = 1..d$ has a characteristic length-scale ℓ_i, and the process itself has a signal variance σ_f^2 and noise variance σ_n^2. These parameters can be optimized using a pure Maximum Likelihood (ML) criterion that depends only on data likelihood, or using a Maximum A Posteriori (MAP)

criterion that is a combination of data likelihood and a prior on parameters.

Likelihood

Recall that the likelihood function for a Gaussian process with prior mean function $\mu(\mathbf{x})$ and kernel k is given by

$$\log p(F_{\mathbf{x}} = \mathbf{f}) = -\frac{1}{2}(\mathbf{f}-\boldsymbol{\mu})^{\mathsf{T}}\,\mathrm{K}^{-1}\,(\mathbf{f}-\boldsymbol{\mu}) - \frac{1}{2}\log|K| - \frac{N}{2}\log 2\pi \tag{3.1}$$

where $\mathrm{K} = k(\mathbf{x},\mathbf{x}) + \sigma_n^2\mathrm{I}$. In the above equation, $\mathbf{x}$, $\mathbf{f}$, and N are completely determined by the data, whereas σ_n^2, $\boldsymbol{\mu}$ and the ℓ_i and σ_f^2 that parameterize k are free model parameters. In many machine learning applications, $\boldsymbol{\mu}$ is fixed at zero; however for our optimization application we would like to be able to model functions that have typical values far from zero without having to greatly increase the signal variance σ_f^2 to account for these. In other words, we want the model to be invariant to additive scaling. To this end, we will always choose the best *constant* $\boldsymbol{\mu}$ for our data and substitute it into the likelihood equation, effectively removing $\boldsymbol{\mu}$ as a parameter by fixing its form to maximize likelihood. We can find this most likely $\boldsymbol{\mu}$ by defining $\mu(\mathbf{x}) = \mu_c$, implying $\boldsymbol{\mu} = \mathbf{1}\cdot\mu_c$. We then take the derivative with respect to the constant μ_c and set it to zero, which gives the maximum likelihood constant prior mean $\boldsymbol{\mu}^*$

$$\begin{aligned}
\log p(F_{\mathbf{x}} = \mathbf{f}) &= -\frac{1}{2}(\mathbf{f}-\boldsymbol{\mu}^*)^{\mathsf{T}}\mathrm{K}^{-1}(\mathbf{f}-\boldsymbol{\mu}^*) - \frac{1}{2}\log|\mathrm{K}| - \frac{N}{2}\log 2\pi \\
\boldsymbol{\mu}^* &= \left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}\right)\mathbf{1}
\end{aligned}$$

where $\mathbf{1}$ is the vector of all ones. Furthermore, since

$$\begin{aligned}
(\mathbf{f}-\boldsymbol{\mu}^*)^{\mathsf{T}}\mathrm{K}^{-1}(\mathbf{f}-\boldsymbol{\mu}^*) &= \mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f} - 2\mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\boldsymbol{\mu}^* + \boldsymbol{\mu}^{*\mathsf{T}}\mathrm{K}^{-1}\boldsymbol{\mu}^* \\
&= \mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f} - 2\mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}\left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}\right) + \mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}\left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}\right)^2 \\
&= \mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f} + \frac{-2\mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}\,(\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}) + (\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f})^2}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}} \\
&= \mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f} - \frac{(\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f})^2}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}
\end{aligned}$$

it follows that if we assume we will always use $\boldsymbol{\mu}^*$ for the mean, the likelihood function can be written

$$\log p(F_{\mathbf{x}} = \mathbf{f}) = -\frac{1}{2}\mathbf{f}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f} - \frac{1}{2}\log|\mathrm{K}| - \frac{N}{2}\log 2\pi - \frac{1}{2}\frac{(\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f})^2}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}} \tag{3.2}$$

The first three terms of this expression represent the likelihood assuming $\boldsymbol{\mu} = \mathbf{0}$, and the last term accounts for our maximizing to find $\boldsymbol{\mu}^*$. Therefore, when we take a partial derivative

with respect to a kernel parameter θ, we can use the usual form [36] of the likelihood gradient plus an extra term to account for our optimization of $\boldsymbol{\mu}$. The derivative the last term above term is

$$\begin{aligned}\frac{\partial}{\partial\theta}\frac{(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f})^2}{\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1}} &= \frac{2(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1})(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f})(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\frac{\partial \mathrm{K}}{\partial\theta}\mathrm{K}^{-1}\mathbf{f}) - (\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f})^2(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\frac{\partial \mathrm{K}}{\partial\theta}\mathrm{K}^{-1}\mathbf{1})}{(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1})^2} \\ &= \frac{1}{(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1})^2}\cdot \\ &\quad \left(2(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1})(\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f})\,\mathbf{f}^\mathsf{T}\mathrm{K}^{-1} - (\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f})^2\,\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\right)\cdot\frac{\partial \mathrm{K}}{\partial\theta}\cdot(\mathrm{K}^{-1}\mathbf{1})\end{aligned}$$

so if we define $\boldsymbol{\alpha} = \mathrm{K}^{-1}\mathbf{f}$, and $\boldsymbol{\gamma} = \mathrm{K}^{-1}\mathbf{1}$, the derivative of the likelihood function is

$$\begin{aligned}\frac{\partial}{\partial\theta}\log p(F_\mathbf{x} = \mathbf{f}) &= \frac{1}{2}\operatorname{tr}\left[(\boldsymbol{\alpha}\boldsymbol{\alpha}^T - \mathrm{K}^{-1})\frac{\partial \mathrm{K}}{\partial\theta}\right] \\ &\quad -\frac{1}{2}\frac{1}{(\boldsymbol{\gamma}^\mathsf{T}\mathbf{1})^2}\left(2(\boldsymbol{\gamma}^\mathsf{T}\mathbf{1})(\boldsymbol{\alpha}^\mathsf{T}\mathbf{1})\,\boldsymbol{\alpha}^\mathsf{T} - (\boldsymbol{\alpha}^\mathsf{T}\mathbf{1})^2\,\boldsymbol{\gamma}^\mathsf{T}\right)\cdot\frac{\partial \mathrm{K}}{\partial\theta}\cdot(\boldsymbol{\gamma}) \quad (3.3)\end{aligned}$$

Note that we only use kernels (Matérn, squared exponential) for which $\partial k(x,z)/\partial\theta$ exists for all x and z in the domain of f, and for all θ we are interested in.

Priors

Previously, consternation has been expressed about using maximum likelihood to determine the length-scales of Gaussian processes when there are few data points. Indeed, we will see that using small amounts of data, optimization procedures using maximum likelihood models initially perform significantly worse than the uniform random strategy. When data are sparse, the likelihood function can become very flat along certain dimensions, or worse yet, become monotonic increasing in certain directions, causing parameters to head toward positive or negative infinity. This can cause a number of problems including producing a kernel matrix that is numerically ill-conditioned, and models that practitioners find unappealing.

Previous approaches (Jones et al. [19], Sasena [39]) to Gaussian process optimization have included an initial pre-acquisition of points in a Latin hypercube design. This initial design is *not* in any way dependent on incoming data. In such a design, n points are chosen by first dividing each dimension into n bins. For each dimension, the bins are permuted and dimension i of point j is drawn uniformly from within the jth bin in the ith permutation. The important result of this is that we are very unlikely to have points where $x_j = z_j$, i.e. we will not observe pairs of points that are parallel to one of the coordinate axes. This arrangement is particularly unhelpful when trying to learn axis-aligned processes. Figure 3.1 shows the result of using maximum likelihood on two data sets that are identical except that

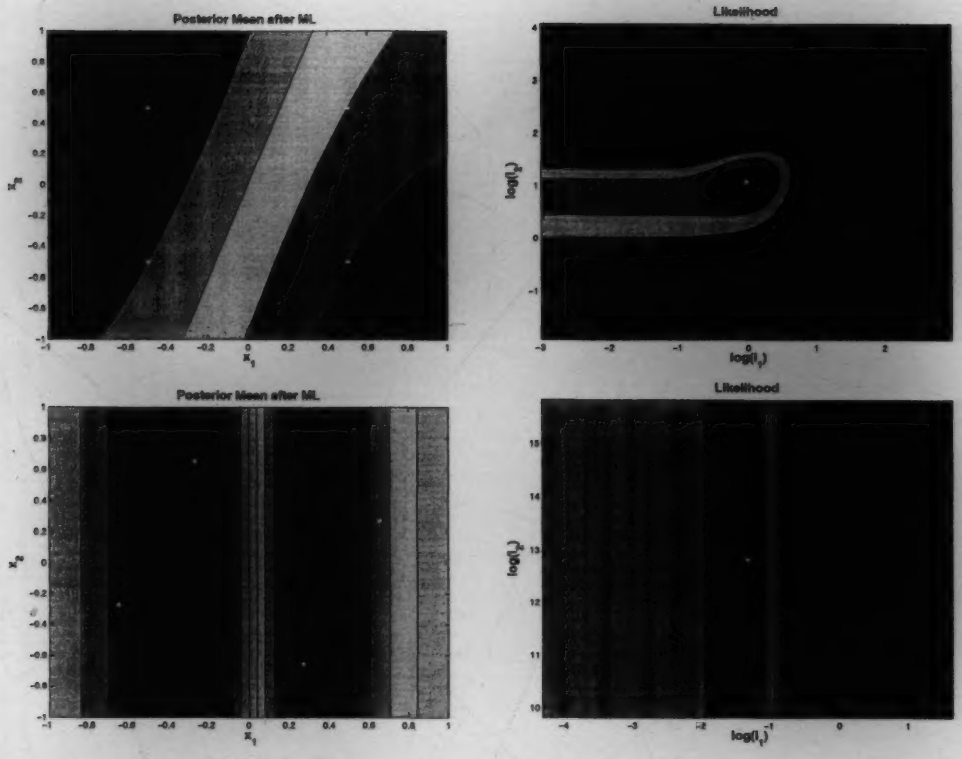


Figure 3.1: Two GPs, one with axis-aligned data and one with non-aligned data. Data points are indicated in the posterior mean plots by small $*$. From the lower-left heading clockwise, the function values are $-0.5, -1.0, 0.5, 1.0$ in both cases. Maxima are indicated in the likelihood plots with the same marker.

one is aligned with the coordinate axes and one is not. Note that in these examples σ_n is fixed at a very small value, a typical practice in the literature [39]. The spacing between data points and the corresponding observation values remain the same; we have simply rotated the points clockwise in the domain by $\pi/8$. Note that in the axis-aligned case, the likelihood function is unimodal, and the maximum likelihood length-scales are both moderate at $(\ell_1, \ell_2) = (0.98, 2.96)$.

In the rotated example, the likelihood function is unimodal along ℓ_1, but monotonic increasing in ℓ_2. The maximum likelihood function found by the hill-climber for this example is $(\ell_1, \ell_2) = (0.260, 371015.82)$, and ℓ_2 is only finite because the function asymptotes (fortunately) so that eventually the gradient becomes sufficiently small for the hill-climber to quit. This asymptotic behaviour illustrates another important point: At $\ell_2 = 371015.82$, the data are only about 1.005 times more likely than at $\ell_2 = 1.0$, for example. However, since the likelihood function does not have *a priori* preferences on length-scales, ℓ_2 is increased drastically to achieve this tiny improvement. The effect of this very large ℓ_2 causes the model to explain the variation in the function observations using ℓ_1 only—this is visible in the plot of the posterior mean, which shows that $\mu([x_1\ x_2]^\mathsf{T})$ does not vary with x_2 over the domain. In the axis-aligned data, however, there is strong evidence that the data cannot be explained by one dimension alone. Take the rightmost two points, which are $w = (0.5, -0.5)$ and $z = (0.5, 0.5)$. In this example, $f(w) = 0.5$ and $f(z) = 1.0$. Since w and z share the same first coordinate, their difference in function value must be explained by a finite ℓ_2. A similar impact on ℓ_1 is produced by points in the square that share the same second coordinate.

This example is not meant to imply that if we have axis-aligned data then the likelihood function will be unimodal, but to illustrate that when data are grid-aligned, maximizing likelihood will not make a length-scales go to infinity, unless of course no variation is ever observed along that direction.

It also illustrates the point that maximizing likelihood to chose parameters can produce counterintuitive results, particularly with small amounts of data. In the example shown, one variable is effectively removed from consideration. It is reasonable to assume that, *a priori*, one assumes that all the variables for a given problem will have some importance; if this were not true it seems those variable would never have been included in the first place.

Finally, it is strong evidence for not using the typical Latin hypercube pre-acquisition, since this approach will specifically choose points so that they are *not* grid-aligned, thus leaving us open to the problems illustrated here. Typically in such designs, $10 \cdot d$ points

are used. We will see from later experiments that this strategy is unnecessary and will attempt to avoid the expense of pre-acquisition altogether, taking the view that if we are unhappy with the models obtained by maximum likelihood, then our objective function is wrong and we should encode our ideas about appropriate models in a prior. We investigate two types of priors, one a simple log-normal prior, and another based on the expected Euler characteristic of excursion sets.[1]

Independent Log-Normal Prior This prior is designed simply to prevent length-scales from getting very large or very small. For the independent log-normal (ILN) prior, we place a normal distribution with mean 0 and standard deviation 10 on the *logarithm* of each of the length-scales. This is a very vague prior; for example it asserts that there is a 95% probability of finding a length-scale between 7.18×10^{-8} and 1.39×10^{7}. We simply add the logs of these probabilities to the likelihood function (3.1) to obtain our new objective

$$\begin{aligned} \log p(F_{\mathbf{x}} = \mathbf{f}) &= -\frac{1}{2}(\mathbf{f} - \boldsymbol{\mu}^*)^{\mathsf{T}} \, \mathrm{K}^{-1} \, (\mathbf{f} - \boldsymbol{\mu}^*) - \frac{1}{2}\log|\mathrm{K}| - \frac{N}{2}\log 2\pi \\ &\quad + \sum_{i=1}^{d}\left(\frac{-\log(\ell_i)^2}{2 \cdot 10^2} - \log 10\sqrt{2\pi}\right) \end{aligned} \tag{3.4}$$

Recall that the ℓ_i in Equation 3.4 are simply the length scales from the kernel as defined in Equations 2.47 and 2.48. These appear explicitly in (3.4) in the prior part, and implicitly in K. In practice we work with $\log(\ell_i)$ for convenience, which makes the derivatives simpler and allows us to use an unconstrained optimizer. To compute the gradient of (3.4) with respect to the log length scales, we therefore simply add the gradient of the likelihood from Equation 3.3 to the gradient of the last term of (3.4), which is

$$\frac{\partial}{\partial \log(\ell_j)}\left[\sum_{i=1}^{d}\left(\frac{-\log(\ell_i)^2}{2 \cdot 10^2} - \log 10\sqrt{2\pi}\right)\right] = \frac{-\log(\ell_j)}{10^2}$$

We will need this gradient for computing MAP estimates of the ℓ_i, discussed below.

Expected Euler Characteristic Prior We also describe a novel prior based on the expected Euler characteristic (EEC), denoted $\mathrm{E}[\chi(\mathcal{A}_u)]$. A definition and novel polynomial time algorithm for this quantity were given in Section 2.3. Recall that the excursion set $\mathcal{A}_u$ of a function above level u consists of all points in the domain where the function exceeds u, and that the Euler characteristic χ of a set is equal to the number of connected components

[1] Note that in this discussion we have not mentioned model averaging. While it would be arguably a more correct approach, we wish to keep computational effort to a minimum, so we will restrict ourselves to using optimized point-estimates of model parameters.

of the set, so long as none of the components have holes in them. The expectation is taken over functions sampled from the Gaussian process. For sufficiently large u, the excursion sets become the union of small disjoint convex sets surrounding the maxima of the function in question and therefore approximates the number of maxima above u. We propose the expected Euler characteristic as a measure of the difficulty of optimizing functions drawn from a particular Gaussian process model—the higher the $\mathrm{E}[\chi(\mathcal{A}_u)]$, the more difficult the optimization.

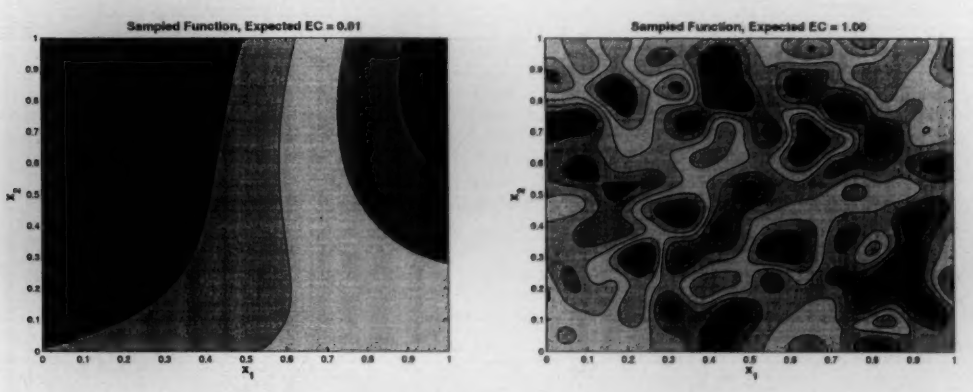


Figure 3.2: Contour plots of two sampled functions, one from a GP with $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 0.01$ and the other with $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 1.0$. For both, $\mu(x) = 0$ and $\sigma_f^2 = 1.0$.

The motivation for the expected Euler characteristic prior is based on the desire for a prior that considers "simple" functions more likely *a priori*. This is somewhat true of the independent log normal prior discussed above, but only for functions of few dimensions. For example, a two-dimensional process with log length-scale parameters $\log(\boldsymbol{\ell}) = [0, 0]$ and a ten-dimensional process with parameters $\log(\boldsymbol{\ell}) = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$ are both at the mode of the independent log-normal prior we use, and are both therefore most preferable *a priori*. However, the two-dimensional process is much simpler, both intuitively and as measured by expected Euler characteristic. For the two-dimensional process, $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 0.0070$, whereas for the ten-dimensional process $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 1.0769$. Figure 3.2 shows sampled functions from processes where $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 0.01$ and $\mathrm{E}[\chi(\mathcal{A}_{3.0})] = 1.0$, and the difference is quite striking.

Our novel prior should place most of its mass between these two extremes of complexity. To that end, we use a normal prior over $\mathrm{E}[\chi(\mathcal{A}_{3.0})]$ that places about 97% of its mass between 0.0 and 0.5 by using parameters $\mu_{\mathrm{eec}} = 0.175$ and $\sigma_{\mathrm{eec}} = 0.0917$. Again, we simply

add this penalty to the likelihood function to get

$$\log p(F_{\mathbf{x}} = \mathbf{f}) = -\frac{1}{2}(\mathbf{f} - \boldsymbol{\mu}^*)^\top \mathrm{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}^*) - \frac{1}{2}\log|K| - \frac{N}{2}\log 2\pi - \frac{(\mathrm{E}[\chi(\mathcal{A}_{3.0})] - \mu_{\mathrm{eec}})^2}{2\sigma_{\mathrm{eec}}^2} - \log \sigma_{\mathrm{eec}}\sqrt{2\pi} \quad (3.5)$$

and again, the length scales from the kernels defined in Equations 2.47 and 2.48 will allow us to compute both the log likelihood and its gradient, as well as $\mathrm{E}[\chi(\mathcal{A}_{3.0})]$ and its gradient, as detailed at the end of Section 2.3.4.

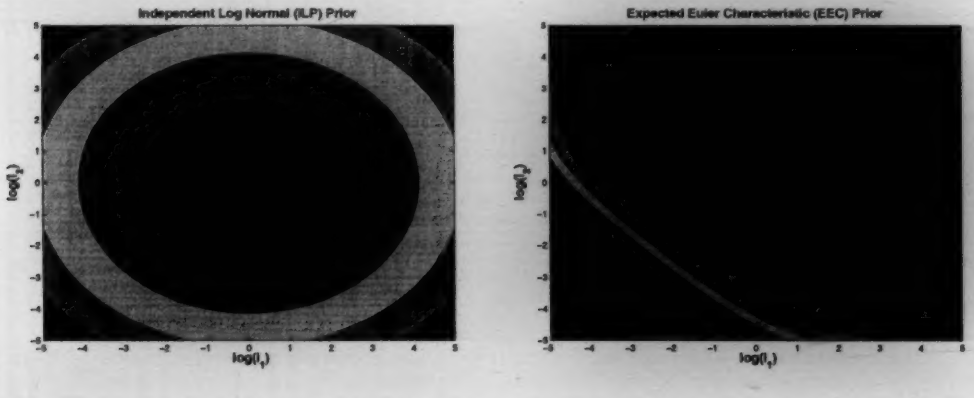


Figure 3.3: Contour plots of the independent log-normal prior and the expected Euler characteristic prior used in this work.

Figure 3.3 shows both the ILN prior and the EEC prior. While both priors prefer log length-scales near the origin, the EEC prior is also willing to allow one length scale to become shorter if the other is made longer. This is because if we fix the number of dimensions of the process, there are sub-manifolds of length-scales that produce processes with the same EEC. The contour lines of the EEC prior in Figure 3.3 show some examples of these sub-manifolds for two-dimensional processes; in d dimensions, these manifolds will be $d-1$ dimensional.

Examples of the impact of these priors on a likelihood function are shown in Figure 3.4 and Figure 3.5. In the left column are the posterior mean functions produced when the various criteria are used for parameter learning, and the right column illustrates the function that was maximized to arrive at these posterior means. One can see that the priors can significantly influence the outcome of parameter learning when there are only a few data points.

Maximization Procedure These two objective functions give us a mechanism for learning the relevant kernel parameters (i.e. length scales) from data. To do so, we simply

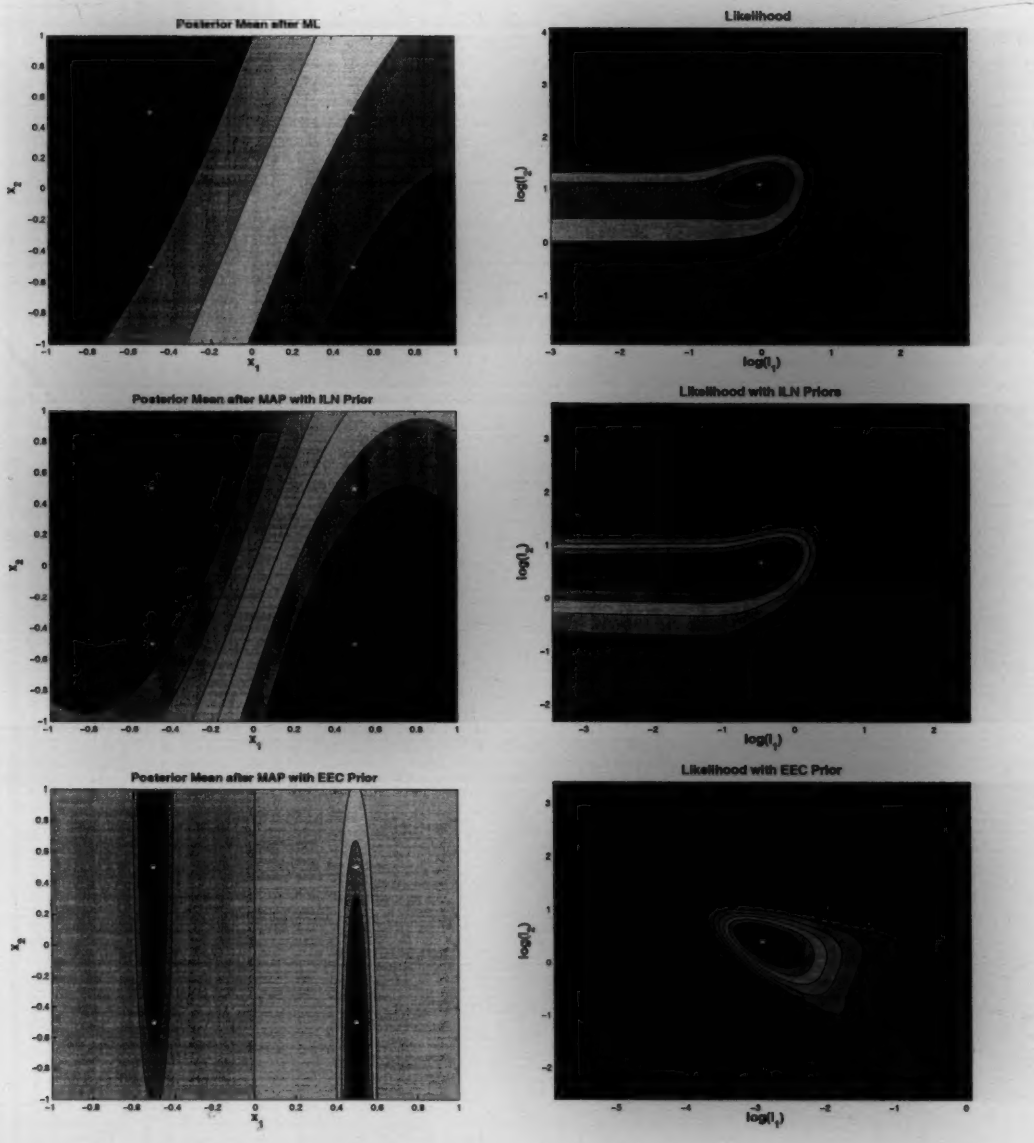

Posterior Mean after ML
Likelihood
Posterior Mean after MAP with ILN Prior
Likelihood with ILN Priors
Posterior Mean after MAP with EEC Prior
Likelihood with EEC Prior

Figure 3.4: Gaussian process with axis-aligned data. In the likelihood graphs, the small asterisk indicates the global maximum.

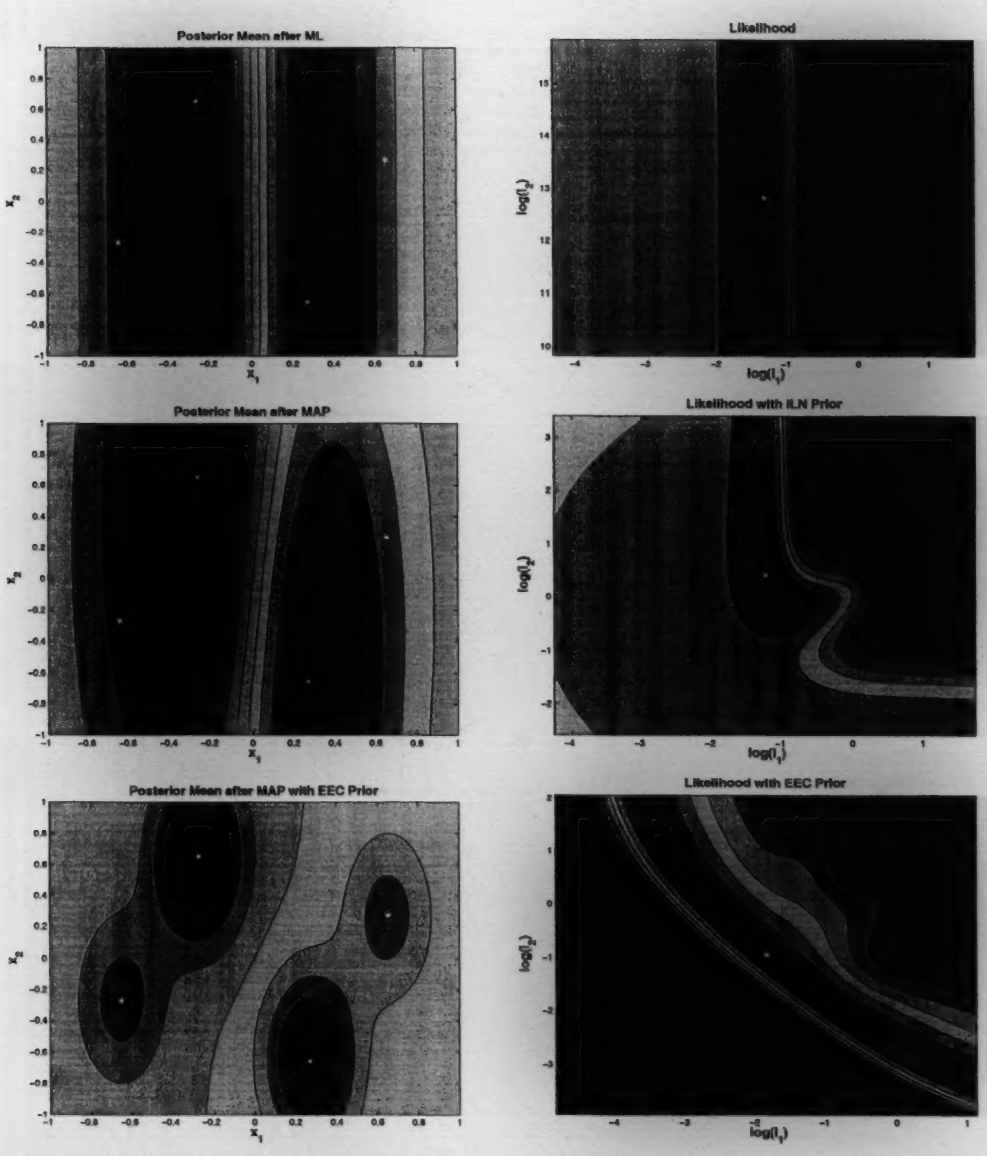

Posterior Mean after ML
Likelihood
Posterior Mean after MAP
Likelihood with ILN Prior
Posterior Mean after MAP with EEC Prior
Likelihood with EEC Prior

Figure 3.5: Gaussian process with non-aligned data. In the likelihood graphs, the small asterisk indicates the global maximum.

maximize (3.4) or (3.5) with respect to the ℓ_i, and then use the resulting parameters in our posterior computations. In order to maximize these criteria, we use an unconstrained quasi-Newton method, starting from the mode of the prior. In particular, we use the BFGS method as implemented in Matlab 7.3. The method is provided with the value and gradient of the objective function, which we compute analytically as shown above.

3.2 Acquisition Criteria

As mentioned earlier; there are two acquisition criteria currently in use for global optimization using Gaussian processes. The Maximum Expected Improvement (MEI) criterion, which selects the point at the maximum of the Expected Improvement (EI) function,

$$\begin{aligned} \mathrm{EI}(x,\xi) &= \mathrm{E}[(F_x - (\mu_{\max} + \xi))_+] && (3.6)\\ &= (\mu(x) - (\mu_{\max} + \xi))\Phi\left(\frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)}\right) && (3.7)\\ &\quad + \sigma(x)\phi\left(\frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)}\right) && (3.8) \end{aligned}$$

and the Maximum Probability of Improvement (MPI) criterion which selects the point at the maximum of the Probability of Improvement (PI) function,

$$\begin{aligned} \mathrm{PI}(x,\xi) &= P[F_x \geq \mu_{\max} + \xi] && (3.9)\\ &= 1 - \Phi\left(\frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)}\right) && (3.10)\\ &\equiv \left(\frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)}\right) && (3.11) \end{aligned}$$

We use $\equiv$ to indicate that the function given by Equation 3.11 is equivalent to the MPI criterion, since removing the Φ and negating is a monotonic transformation and therefore the maxima and minima do not change location. Recall that $\mu(x)$ and $\sigma(x)$ are the posterior mean and standard deviation of F_x, and $\mu_{\max} = \max_x \mu(x)$.

Both of these criteria prefer points with higher posterior mean and higher posterior variance, but the tradeoff between these two quantities is qualitatively different for the different criteria. This is tradeoff is controlled by the parameter $\xi > 0$. Figure 3.6 shows this tradeoff graphically. For PI, the tradeoff is straightforward; for a fixed σ, PI increases linearly in μ. For fixed μ, it increases like $-1/\sigma$ since the numerator in (3.11) is always negative by definition. This brings up an important point: PI is bounded above by 0, and PI $= 0$ is achieved only when $\xi = 0$ and $\mu(x) = \mu_{\max}$, regardless of the value of $\sigma(x)$. It is

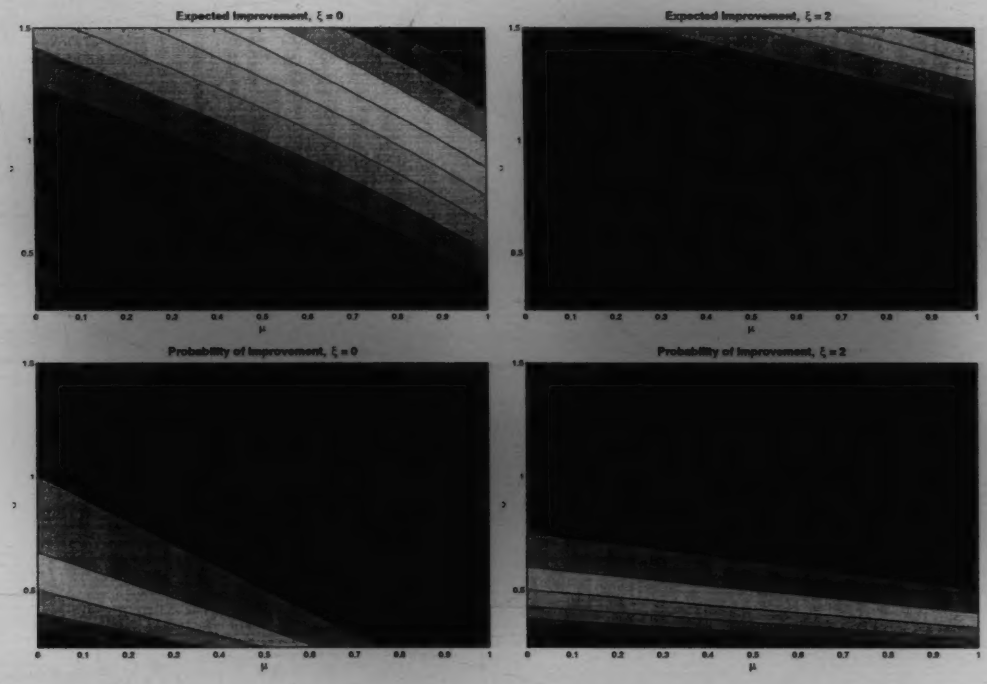


Figure 3.6: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maxima of each of these is in the top-right corner. Note that for higher ξ, increasing σ has a greater impact than increasing μ, particularly for PI.

easy to see therefore that we can can make MPI purely "greedy" by setting $\xi = 0$. In this case, using MPI will always acquire the maximum point of the posterior mean.

On the other hand, EI is bounded *below* by 0 by definition, and MEI $= 0$ is only achieved $\xi = 0$ and $\mu(x) = \mu_{\max}$ *and* $\sigma(x) = 0$. Therefore MEI is not "greedy" even when $\xi = 0$ and will not always select the point where $\mu(x) = \mu_{\max}$. Furthermore, the tradeoff MEI makes between μ and σ is more complicated than that of MPI, although the preference for higher μ and σ is maintained.

The plots in Figure 3.6 immediately demonstrate what kind of point we would choose if we could select μ and σ; however we are only allowed to select the domain point x which determines these values. Acquisition criterion optimization can be though of as a search over x in the domain (as it is usually formulated) or as a constrained optimization over μ and σ, with the feasible set being those (μ, σ) pairs that can be found in our Gaussian process.

Heuristics for Criterion Optimization This brings up an important point: In order to use these criteria, we must maximize them over the domain of f, and both EI and PI can be complicated, multi-modal functions. For processes with many dimensions, it will be impossible to guarantee finding the global maximum. Nevertheless, we can take advantage properties of these functions to at least find points where the criteria are large, if not maximal.

Both criteria can be written as functions of

$$\mathcal{Z} = \frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)} \tag{3.12}$$

With this definition we have,

$$\text{EI} = (\mu(x) - (\mu_{\max} + \xi))\Phi(\mathcal{Z}) + \sigma(x)\phi(\mathcal{Z}) \tag{3.13}$$

$$\text{PI} = \mathcal{Z} \tag{3.14}$$

Each of these functions has properties that make them difficult to maximize in certain situations.

For areas where $\mathcal{Z}$ is small, EI in and its gradient become numerically zero because Φ and ϕ fall off exponentially as we decrease $\mathcal{Z}$. In Matlab 7.3, for example, these functions[2] are identically zero for $\mathcal{Z} \leq -40$. Perhaps the worst side-effect of this is that if we want to optimize EI by hill-climbing, if we start at a point that is very poor then we cannot make any progress.

For MPI, the problem is the opposite. Particularly when ξ is small, maxima of $\mathcal{Z}$ can become very flat since it is bounded *above* by 0. From a practical standpoint, the problem is less severe since even if we fail to accurately maximize PI we can definitely make progress from poor points because the function and its gradient are not bounded below, so we can always find a search direction that improves PI from such points.

In view of these properties, we use the following heuristic for finding the MPI point: Since it is easy to make progress from poor points, we evaluate PI at a small number of uniform random domain points (we used 100) and run a quasi-Newton hill-climber from the best of these (we used 5). There are certainly more intelligent and complicated heuristics that could be used here; however we have found that even with this simple approach the PI criterion performs well in experiments; see Chapter 5 for details.

Since EI is sometimes difficult to maximize from very poor points, as described above, we modify the heuristic slightly for this criterion. Again we use a small number of uniform

[2]They are named `normcdf` and `normpdf`, respectively, in this software.

random domain points (again 100) and run a quasi-Newton hill-climber from the best of these (again 5). However, we also always run the hill-climber from the point found by MPI, since this hopefully has a high $\mathcal{Z}$ and therefore a nonzero gradient, allowing us to improve the point further.

Invariant Improvement Earlier in the chapter, we mentioned the desire to make our optimization procedure invariant to any additive shift, motivated by the knowledge that such scaling does not change the location of optima of the target function. We achieve this by choosing the maximum likelihood constant mean: When we do this, the EI and PI functions become invariant to any additive constant in the data, which we will show shortly. We then present novel modifications to the MEI and MPI criteria that make them invariant to multiplicative scaling also.

First, recall the likelihood function we are using is given by

$$\begin{aligned} \log p(F_{\mathbf{x}} = \mathbf{f}) &= -\frac{1}{2}(\mathbf{f} - \boldsymbol{\mu}^*)^{\mathsf{T}}\mathrm{K}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - \frac{1}{2}\log|\mathrm{K}| - \frac{N}{2}\log 2\pi \\ \boldsymbol{\mu}^* &= \left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}\right)\mathbf{1} \end{aligned}$$

and note that if we shift the data by a constant c, we have

$$\begin{aligned} \boldsymbol{\mu}^*[c] &= \left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}(\mathbf{f} + c\cdot\mathbf{1})}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}}\right)\mathbf{1} \\ &= \left(\frac{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^{\mathsf{T}}\mathrm{K}^{-1}\mathbf{1}} + c\right)\mathbf{1} \\ &= \boldsymbol{\mu}^* + c\cdot\mathbf{1} \end{aligned}$$

therefore shifting the data by c shifts the maximum likelihood mean by c, which means that $(\mathbf{f} - \boldsymbol{\mu}^*)$ remains the same, as does the rest of the likelihood function. Therefore if we are maximizing likelihood and we shift the data additively, estimates of all the other parameters remain the same, and the posterior mean

$$\mu(x) = \mu^*(x) + k(x, \mathbf{z})\,\mathrm{K}^{-1}\,(\mathbf{f} - \boldsymbol{\mu}^*)$$

is also shifted by c. The posterior variance function, which does not involve the observed data but only the locations of observations, does not change. Now, note that

$$\mathcal{Z} = \frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)}$$

It is clear now that if we shift the data by c then $\mu(x)$ and $\mu_{\max}$ also shift by c, so we have

$$\begin{aligned} \mathcal{Z}[c] &= \frac{\mu(x) + c - (\mu_{\max} + c + \xi)}{\sigma(x)} \\ &= \frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)} \\ &= \mathcal{Z} \end{aligned} \tag{3.15}$$

so $\mathcal{Z}$ (and PI) are unchanged by an additive shift in the data. It is also immediate that the shifted expected improvement criterion

$$\mathrm{EI}[c] = (\mu(x) + c - (\mu_{\max} + c + \xi))\Phi(\mathcal{Z}) + \sigma(x)\phi(\mathcal{Z}) \tag{3.16}$$

$$= (\mu(x) - (\mu_{\max} + \xi))\Phi(\mathcal{Z}) + \sigma(x)\phi(\mathcal{Z}) \tag{3.17}$$

$$= \mathrm{EI}[c] \tag{3.18}$$

is unaffected. Therefore a global optimization strategy based on a maximum likelihood estimate of the constant mean that uses either of these criteria will select the same points to acquire regardless of any additive shift in the data.

We now consider the effect of a multiplicative scaling s of the data. First, it is easy to see what happens to the original $\boldsymbol{\mu}^*$:

$$\begin{aligned} \boldsymbol{\mu}^*[s] &= \left(\frac{\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}(s \cdot \mathbf{f})}{\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1}}\right)\mathbf{1} \\ &= s \cdot \left(\frac{\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{f}}{\mathbf{1}^\mathsf{T}\mathrm{K}^{-1}\mathbf{1}}\right)\mathbf{1} \\ &= s \cdot \boldsymbol{\mu}^* \end{aligned}$$

Unsurprisingly, $\boldsymbol{\mu}^*$ is scaled by the amount s. Next we address what happens to the maximum likelihood estimate of the signal variance σ_f^2, which we denote $\hat{\sigma}_f^2$. For convenience, we define the following quantities:

$$r(x, z) = \frac{1}{\sigma_f^2} k(x, z) \tag{3.19}$$

$$\mathrm{R} = \frac{1}{\sigma_f^2} k(\mathbf{x}, \mathbf{x}) \tag{3.20}$$

$$= \frac{1}{\sigma_f^2}\mathrm{K} \tag{3.21}$$

Note that k depends linearly on σ_f, so r does not depend on σ_f. Whereas k gives the covariance between domain points, and r gives the correlation. Using these definitions, we can write the likelihood function as

$$\log p(F_\mathbf{x} = \mathbf{f}) = -\frac{1}{2\sigma_f^2}(\mathbf{f} - \boldsymbol{\mu}^*)^\mathsf{T}\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - \frac{1}{2}\log\sigma_f^2 - \frac{1}{2}\log|\mathrm{R}| - \frac{N}{2}\log 2\pi$$

and, by taking derivatives with respect to σ_f^2 and equating to zero, we can find the maximum likelihood estimate

$$\hat{\sigma}_f^2 = (\mathbf{f} - \boldsymbol{\mu}^*)^\mathsf{T} \mathrm{R}^{-1} (\mathbf{f} - \boldsymbol{\mu}^*) \tag{3.22}$$

We can substitute this back into the likelihood function to get

$$\log p(F_\mathbf{x} = \mathbf{f}) = -\frac{1}{2} - \frac{1}{2}\log \hat{\sigma}_f^2 - \frac{1}{2}\log |\mathrm{R}| - \frac{N}{2}\log 2\pi$$

If we scale the data, we have

$$\begin{aligned} \hat{\sigma}_f^2[s] &= (s\cdot\mathbf{f} - s\cdot\boldsymbol{\mu}^*)^\mathsf{T} \mathrm{R}^{-1} (s\cdot\mathbf{f} - s\cdot\boldsymbol{\mu}^*) && (3.23)\\ &= s^2 \cdot (\mathbf{f} - \boldsymbol{\mu}^*)^\mathsf{T} \mathrm{R}^{-1} (\mathbf{f} - \boldsymbol{\mu}^*) && (3.24)\\ &= s^2 \cdot \hat{\sigma}_f^2 && (3.25) \end{aligned}$$

implying that

$$\begin{aligned} \log p(F_\mathbf{x} = \mathbf{f}) &= -\frac{1}{2} - \frac{1}{2}\hat{\sigma}_f^2[s] - \frac{1}{2}\log |\mathrm{R}| - \frac{N}{2}\log 2\pi \\ &= -\frac{1}{2} - \frac{1}{2}\log s^2 \cdot \hat{\sigma}_f^2 - \frac{1}{2}\log |\mathrm{R}| - \frac{N}{2}\log 2\pi \\ &= -\frac{1}{2} - \frac{1}{2}\log s^2 - \frac{1}{2}\log \hat{\sigma}_f^2 - \frac{1}{2}\log |\mathrm{R}| - \frac{N}{2}\log 2\pi \end{aligned}$$

and since $-(1/2)\ \log s$ is a constant, this objective equivalent to the un-scaled objective. This means that when we scale the data, the *only* effect is to scale $\hat{\sigma}_f^2$ and $\boldsymbol{\mu}^*$, since the other kernel parameters (i.e. length scales) only influence the $-(1/2)\ \log |\mathrm{R}|$ term, which is unaffected by scaling. In other words, the length scales learned from the data vector $\mathbf{f}$ will be identical to those learned from data vector $2\cdot\mathbf{f}$.

We now turn our attention back to the acquisition criteria, starting with the quantity $\mathcal{Z}$, which can be written

$$\begin{aligned} \mathcal{Z} &= \frac{\mu(x) - (\mu_{\max} + \xi)}{\sigma(x)} \\ &= \frac{(\hat{\sigma}_f^2 r(x,\mathbf{z}))\frac{1}{\hat{\sigma}_f^2}\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \xi)}{\sqrt{\hat{\sigma}_f^2 r(x,x) - (\hat{\sigma}_f^2 r(x,\mathbf{z}))\frac{1}{\hat{\sigma}_f^2}\mathrm{R}^{-1}(\hat{\sigma}_f^2 r(\mathbf{z},x))}} \\ &= \frac{1}{\hat{\sigma}_f}\cdot\frac{r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \xi)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1} r(\mathbf{z},x)}} \end{aligned} \tag{3.26}$$

We arrive at Equation 3.26 by substituting the Gaussian process formulae in for $\mu(x)$ and $\sigma(x)$. This illustrates an important point: If the data are scaled by an amount s, then

$$\begin{aligned} \mathcal{Z}[s] &= \frac{1}{s\cdot\hat{\sigma}_f}\cdot\frac{s\cdot r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (s\cdot\mu_{\max} + \xi)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1} r(\mathbf{z},x)}} \\ &= \frac{1}{\hat{\sigma}_f}\cdot\frac{r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \xi/s)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1} r(\mathbf{z},x)}} \end{aligned}$$

Note that because of the lone ξ in the numerator, $\mathcal{Z}$ is *not* invariant to scaling. For example, scaling by a factor $s < 1$ is equivalent to increasing ξ by a factor of $1/s$, meaning that the MPI and MEI criteria will shift in focus toward points of higher variance, even though such a scaling would not change the locations of optima of the target function.

We now present a new alternative to $\mathcal{Z}$ that is scale invariant. Consider the following modified objective, denoted $\mathcal{Z}_r$. The subscript r is intended to indicate that the objective is "relative" to the scale of the target function.

$$\begin{aligned} \mathcal{Z}_r &= \frac{\mu(x) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi)}{\sigma(x)} \\ &= \frac{1}{\hat{\sigma}_f} \cdot \frac{r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi_r)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)}} \end{aligned} \tag{3.27}$$

Suppose we now scale the data $\mathbf{f}$ by an amount s. Then this new objective becomes

$$\begin{aligned} \mathcal{Z}[s] &= \frac{1}{s \cdot \hat{\sigma}_f} \cdot \frac{s \cdot r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (s \cdot \mu_{\max} + s \cdot \hat{\sigma}_f \cdot \xi_r)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)} \\ &= \frac{1}{\hat{\sigma}_f} \cdot \frac{r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi_r)}{\sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)}} \end{aligned}$$

therefore changing the scale of the data does not affect the quantity $\mathcal{Z}_r$.

Furthermore, if we modify MEI in a similar way, we obtain

$$\mathrm{EI}_r = (r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi_r))\Phi(\mathcal{Z}_r) + \sigma(x)\phi(\mathcal{Z}_r) \tag{3.28}$$

$$\begin{aligned} &= (r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi_r))\Phi(\mathcal{Z}_r) \\ &\quad + (\hat{\sigma}_f \cdot \sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)})\phi(\mathcal{Z}_r) \end{aligned} \tag{3.29}$$

and again if we scale the data by s and maximize likelihood, we get

$$\begin{aligned} \mathrm{EI}_r[s] &= (s \cdot r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*) - (s \cdot \mu_{\max} + s \cdot \hat{\sigma}_f \cdot \xi_r))\Phi(\mathcal{Z}_r) \\ &\quad + (s \cdot \hat{\sigma}_f \cdot \sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)})\phi(\mathcal{Z}_r) \end{aligned} \tag{3.30}$$

$$\begin{aligned} &= s \cdot [(r(x,\mathbf{z})\mathrm{R}^{-1}(\mathbf{f} - \boldsymbol{\mu}^*)) - (\mu_{\max} + \hat{\sigma}_f \cdot \xi_r)\Phi(\mathcal{Z}_r) \\ &\quad + (\hat{\sigma}_f \cdot \sqrt{r(x,x) - r(x,\mathbf{z})\mathrm{R}^{-1}r(\mathbf{z},x)})\phi(\mathcal{Z}_r)] \end{aligned} \tag{3.31}$$

$$= s \cdot \mathrm{EI}_r \tag{3.32}$$

Consequently, while EI_r is not invariant to scaling, the effect of scaling the data by s is simply to scale the EI_r function by the same amount. Therefore the optima of EI_r remain the same, and any strategy relying on maximum likelihood estimates $\boldsymbol{\mu}^*$ and $\hat{\sigma}_f$ and using the EI_r criterion to select points will select the same points regardless of any multiplicative

scaling of the target function. Furthermore, the PI_τ and EI_τ functions are also invariant to additive shifts by the same arguments used for PI and EI, simply by re-writing $\xi = \hat{\sigma}_f^2 \cdot \xi_\tau$.

Figures 3.7, 3.8, 3.9, 3.10, 3.11, and 3.12 illustrate the MPI_τ and MEI_τ criteria on six different example GPs. These examples illustrate the type of points that both criteria prefer: Both criteria select points where the posterior mean is high and the posterior variance is high. (Though not illustrated, bear in mind that the posterior variance increases as we move farther from the observed data.) They also illustrate the effect of increasing ξ_τ from 0.01 to 0.7: At the higher level of ξ_τ, both criteria select points that have higher posterior variance and hence are further from the observed data points.

3.3 Summary

We have described in detail the Gaussian process model we use for the remainder of this work, which is based on ARD. However, our model may use either of two proposed priors on length scales, one a simple log-normal prior and another novel prior based on the expected Euler characteristic of excursion sets, made feasible by the novel polynomial time algorithm for computing EEC given in Section 2.3.

We have also given two new acquisition criteria that are based on the existing MEI and MPI criteria, but that are invariant to multiplicative scaling of the objective function. These novel objectives, denoted MEI_τ and MPI_τ to indicate that they are "relative" to the scale of the function, will be empirically investigated in the following chapters.

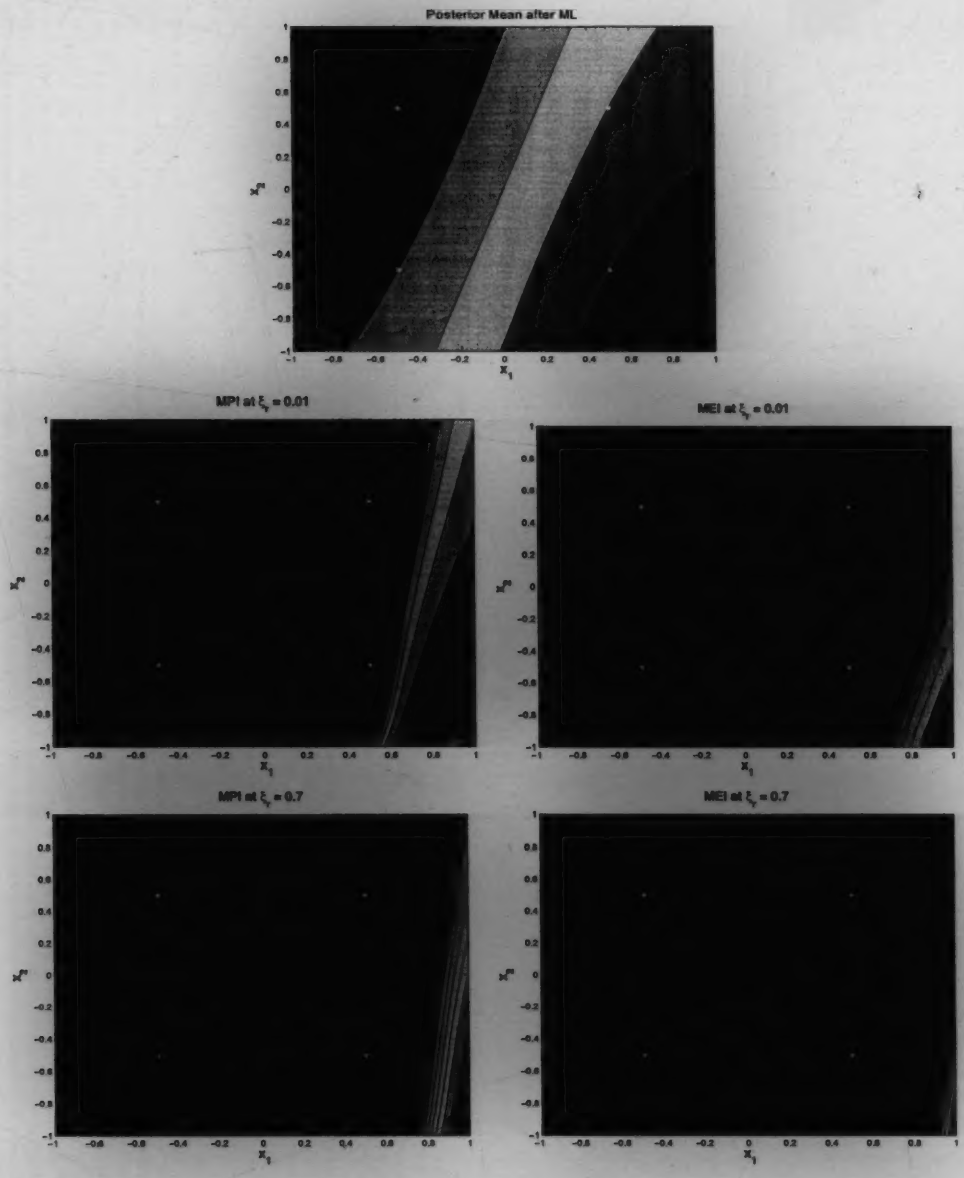


Figure 3.7: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

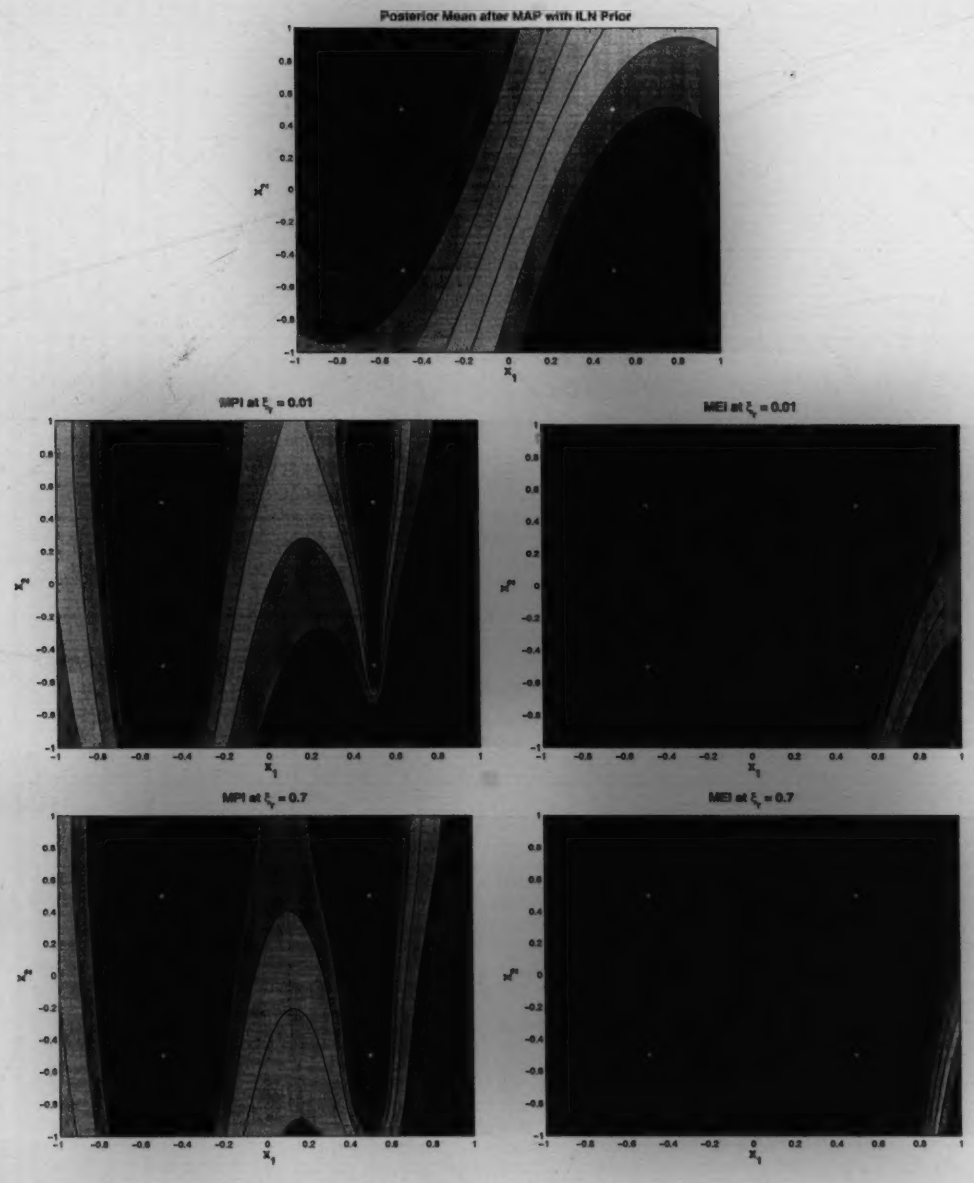


Figure 3.8: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

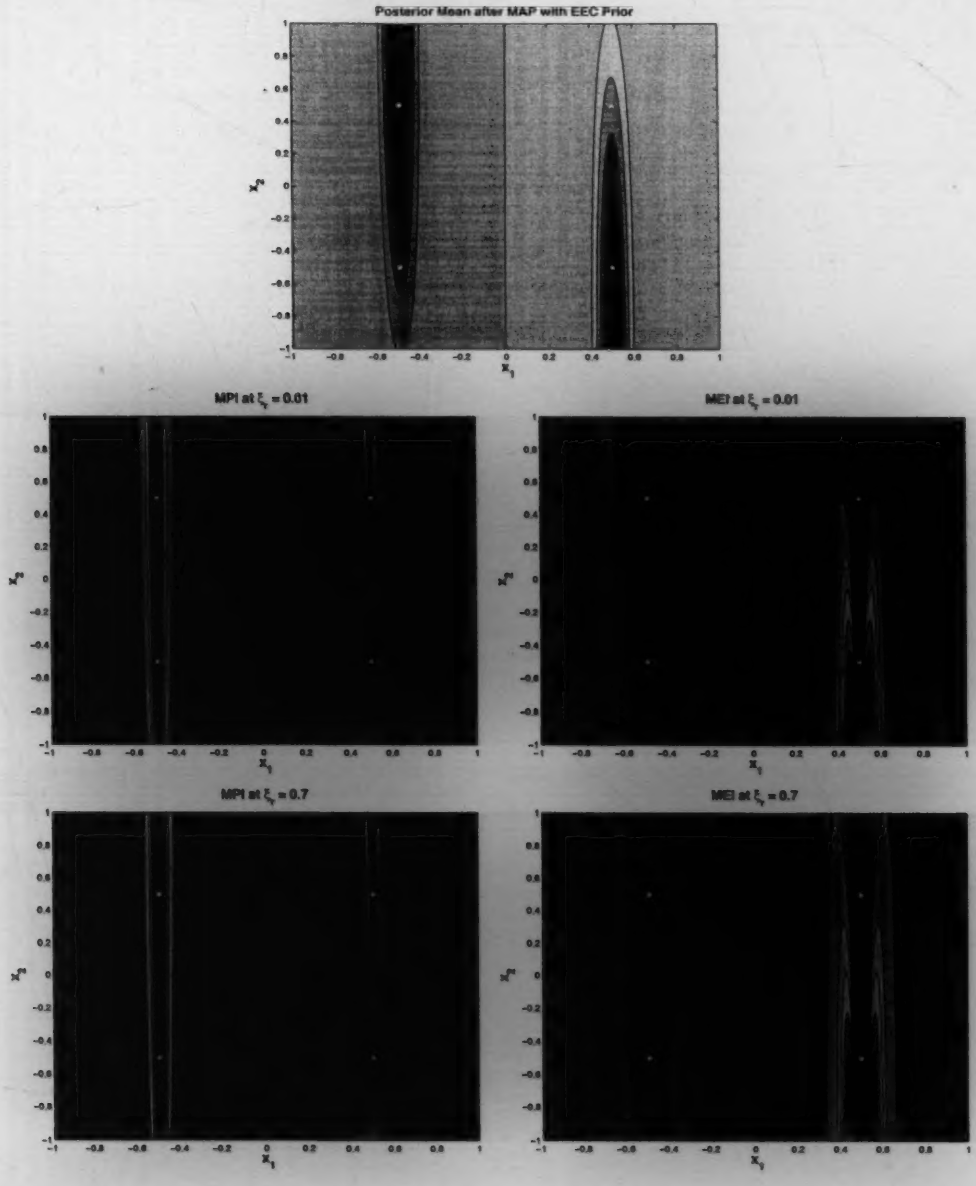


Figure 3.9: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

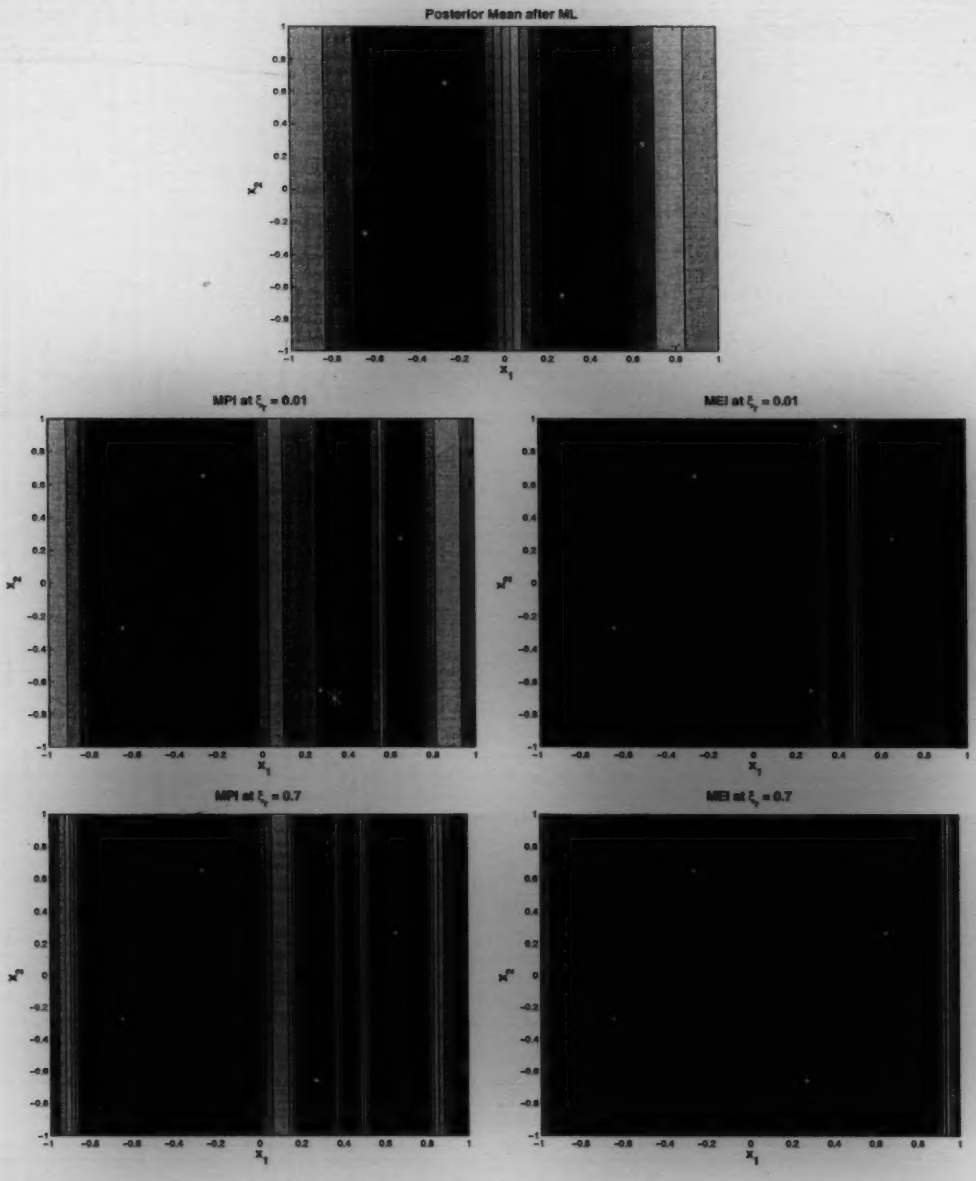


Figure 3.10: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

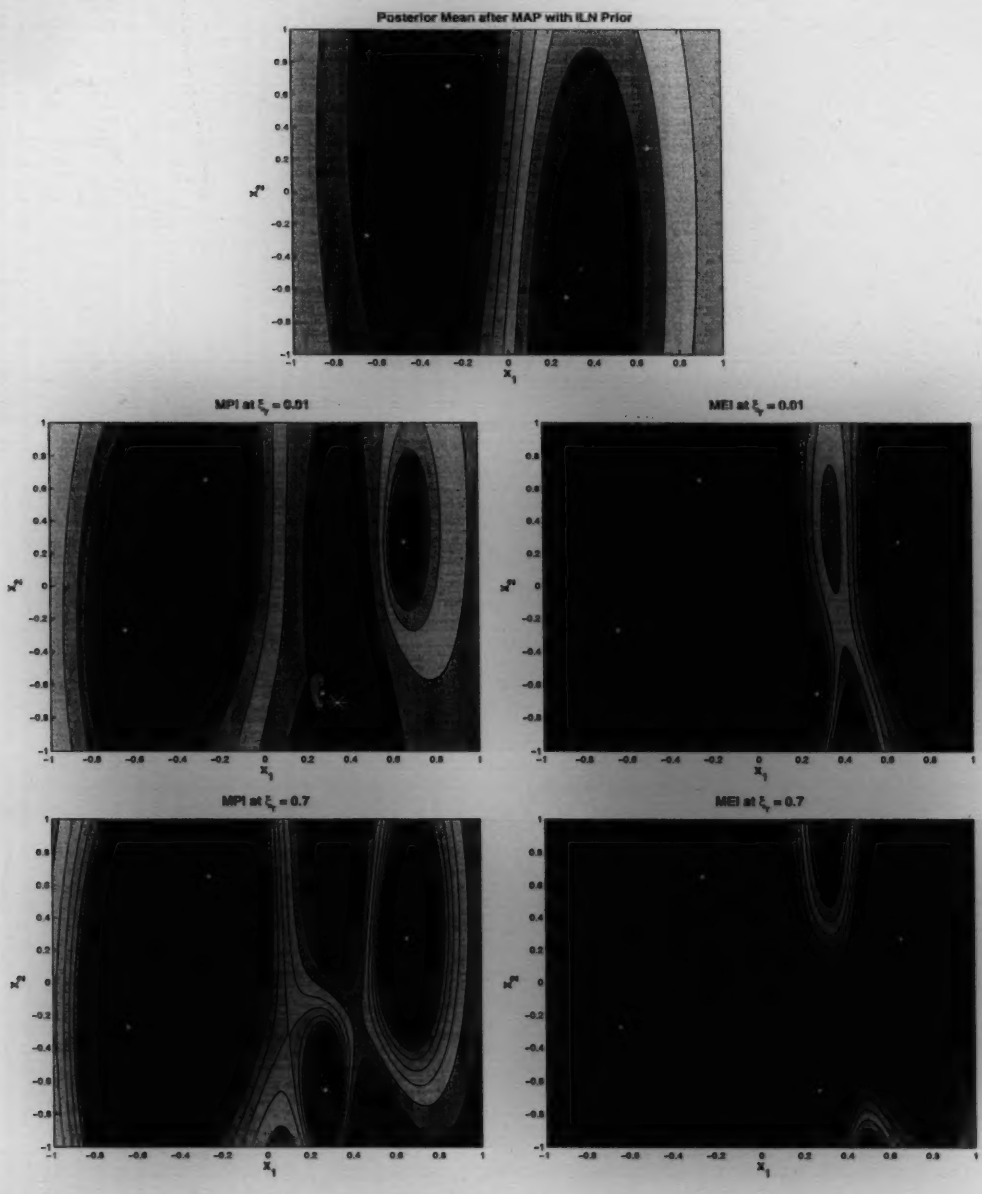


Figure 3.11: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

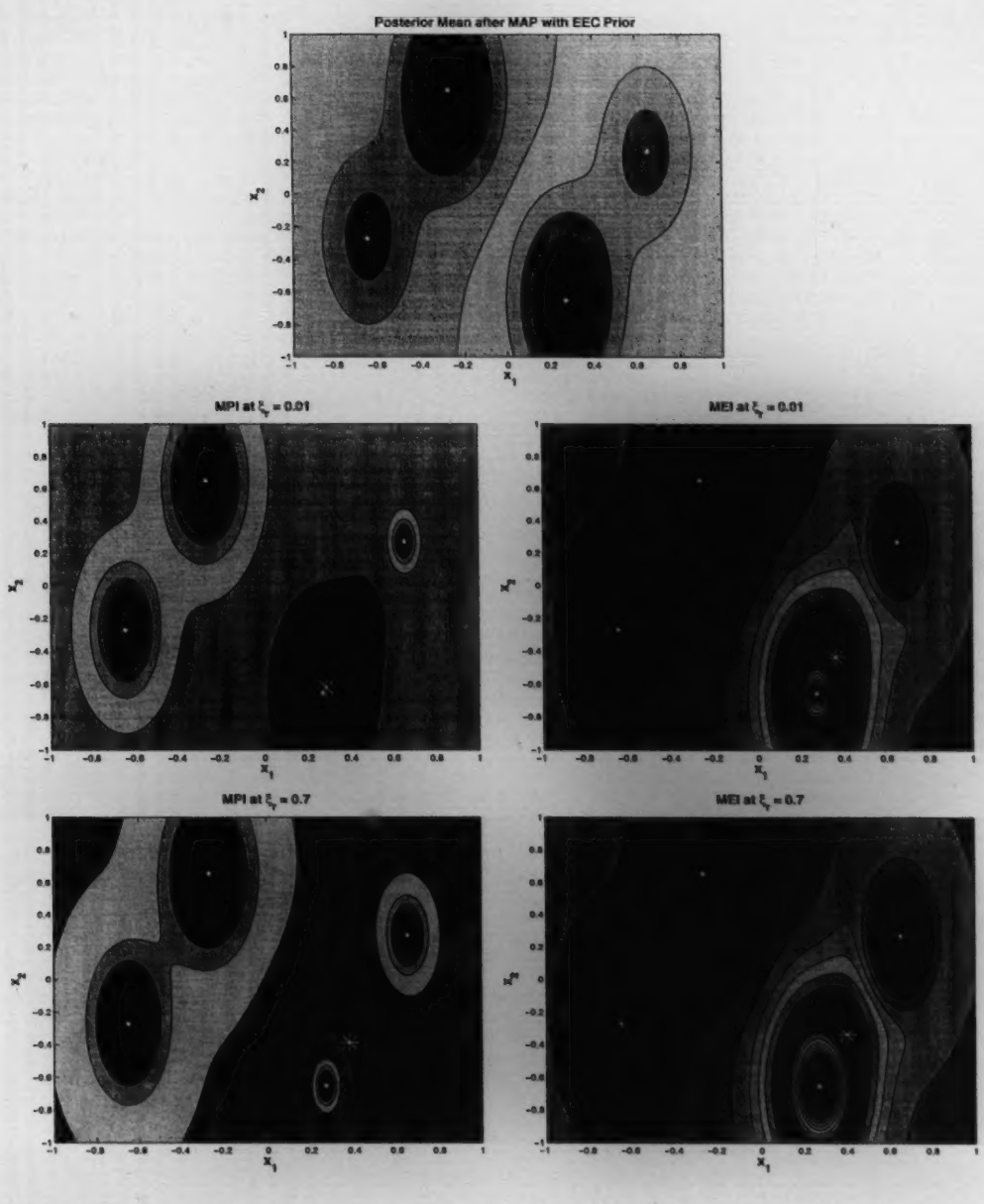


Figure 3.12: The MEI and MPI acquisition criteria. In the contour plots, dark red indicates the highest function values; the maximum of each criterion is marked by a large white asterisk.

Chapter 4

An Experimental Framework for Global Optimization

Our literature review failed to find a single Gaussian process based optimization paper with more than eighteen different example functions. This type of empirical evaluation can be damaging when authors begin to "over-fit" a small suite of objectives that come to be considered "standard test problems". Fortunately, in Bayesian optimization, we are in a good position to avoid this problem.

The MEI and MPI acquisition criteria were constructed as approximations to the Bayes optimal acquisition criterion. That is, they are intended to approximate the best strategy in expectation, *when functions are drawn from the prior.* Since we have the prior, which is supposed to reflect the type of functions we expect to see, it seems natural to draw functions from it to evaluate the performance of the commonly used acquisition criteria in different situations and with different parameters. No such study has ever been done, to the best of our knowledge.

Our experiments are therefore constructed as follows: For each experiment, we fix a *test model*, which is a Gaussian process with fixed parameters and kernel from which we draw multiple *test functions.* For each of these test functions, we construct an *optimization model* which we learn using data acquired from the test function. The information given to the optimization model (kernel, length scales, priors on length scales, derivatives, etc.) will vary between optimization models depending on the properties we wish to investigate.

Furthermore, an empirical examination of the effect of an incorrect model is not difficult to devise. We will simply chose a test model with, for example, a different kernel from the one assumed in the optimization model, which allows us to assess the robustness of different methods and models.

4.1 Models

For the test models, we use Gaussian processes with a constant prior mean of zero and unit signal variance σ_f^2 over a rectangle $\mathcal{I} = [-1, 1]^d$. We do not vary these parameters because the criteria we are assessing, MEI_τ and MPI_τ are invariant to additive shifts and multiplicative scaling under the models we use. Therefore our test models are completely determined by the kernel of the GP and its parameters.

The optimization models are of course over the same domain, however they may or may not use fixed parameters. We investigate the effects of having to learn these parameters using maximum likelihood or MAP optimization as discussed in Chapter 3. We also investigate the effects of using the same or different kernels in the test and optimization models.

4.1.1 Kernels

The kernels we consider are the squared exponential kernel and its relative, the Matérn kernel with $\nu = 3/2$. (We will henceforth refer to this simply as the "Matérn" kernel.) The main difference between these two kernels is in their smoothness. Two examples—one drawn from a process with each type of kernel—are shown in Figure 4.1.

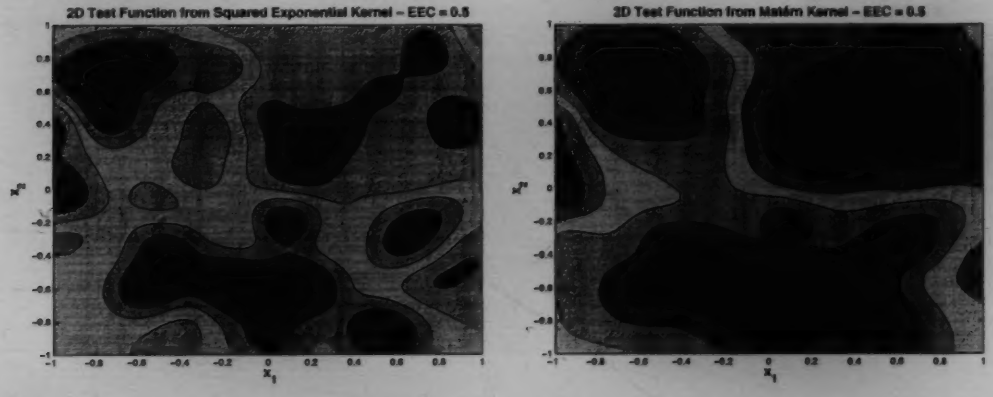


Figure 4.1: Two example test functions, one drawn from a GP with a squared exponential kernel, and the other from a GP with a Matérn kernel.

One way of describing the smoothness of a process is by examining its mean square continuity and differentiability. A Gaussian process is continuous in mean square at a point x^* if

$$\lim_{x \to x^*} \mathrm{E}[|F(x) - F(x^*)|^2] = 0 \tag{4.1}$$

and has a mean square derivative of $d_i(x)$ with respect to x_i at x if

$$\lim_{h \to 0} \mathrm{E}\left[\frac{F(x + h\hat{\mathbf{e}}^i) - F(x)}{h} - d_i(x)\right]^2 = 0 \tag{4.2}$$

where $e^i_j = 1$ if $i = j$, $e^i_j = 0$ if $i \neq j$ [36]. Higher order mean square derivatives are similarly defined. A Gaussian process has mean square derivatives of order k if and only if

$$\lambda_{\mathbf{ij}} = \left.\frac{\partial^{2k} k(\mathbf{s}, \mathbf{t})}{\partial s_{i_1} \partial s_{i_2} \ldots \partial s_{i_k} \partial t_{j_1} \partial t_{j_2} \ldots \partial t_{j_k}}\right|_{\mathbf{s}=\mathbf{t}} \tag{4.3}$$

exists and is finite. This is the matrix of $2k$th order *spectral moments.*

The squared exponential kernel is mean square continuous and infinitely mean square differentiable, whereas he Matérn kernel with $\nu = 3/2$ is mean square continuous but mean square differentiable only once. As such, they represent extremes of this kind of smoothness if we restrict ourselves to differentiable processes. Furthermore, all posterior means and variances remain twice differentiable, since these are expressed as weighted sums of kernel functions, and both of these kernel functions are twice differentiable. This differentiability is required for our approach, since we use hill climbing to optimize the MEI_r and MPI_r criteria as described in Chapter 3.

4.1.2 EEC, Complexity, and Dimensionality

The overall functional form of the kernel tells us these broad characteristics of the resulting process, but the spectral moments can tell us more about process behaviour. In particular, the second order spectral moments

$$\lambda_{ij} = \frac{\partial^2 k(s,t)}{\partial s_i \partial t_j} \tag{4.4}$$

are of interest since they determine the expected Euler characteristic of the process. Recall that, when using the ARD approach, exponential and Matérn kernels are parameterized by a length-scale parameter ℓ_i in each dimension, resulting in axis-scaled isotropic processes. These length-scales appear in the second spectral moments of these kernels as follows:

$$\begin{aligned} \lambda_{ii}^{\text{Matérn}} &= 3\sigma_f^2/\ell_i^2 \\ \lambda_{ii}^{\text{SqExp}} &= \sigma_f^2/\ell_i^2 \end{aligned}$$

These are computed simply using Equation 4.4. Note that in the ARD case, $\lambda_{ij} = 0$ for all $i \neq j$. We use the relationship between λ_{ii} and ℓ_i to compute the expected Euler characteristic of a process, which in turn lets us use a root-finding technique to find length

scales that would give a process a particular EEC. We use this approach to select test models.

Using EEC as our criterion for selecting test models allows us to generate models that have different length scale configurations—even different dimensions—but that are of similar difficulty. We have chosen $\mathrm{E}[\chi(\mathcal{A}_{3\sigma_f})]$ as our quantity to hold constant when generating problems. This is the expected Euler characteristic of the set $\{x : f(x) > 3\sigma_f\}$, i.e. the set of points where the function is three signal standard deviations above the mean. (All EEC values mentioned in this work are in fact $\mathrm{E}[\chi(\mathcal{A}_{3\sigma_f})]$, so we will use "EEC" to mean $\mathrm{E}[\chi(\mathcal{A}_{3\sigma_f})]$, even though it could be more general.) We chose this threshold because it is where the expected Euler characteristic approximates excursion probability—in the region $u \geq 3\sigma_f^2$ we have the property that $\mathrm{E}[\chi(\mathcal{A}_{3\sigma_f})] \approx p(\exists x : f(x) > 3\sigma_f)$ [2], and we hypothesize that the greater chance a process has of achieving such a high value, the more difficult a test function drawn from it will be to optimize. The empirical results that follow indicate that this is at least somewhat true; even if we increase the number of dimensions significantly we can still achieve good success if the EEC is kept in check. If we do not do so, it becomes impossible to even draw functions from the test model because we cannot use enough points to accurately reflect the shape of sampled function. If we sample 100 points, say, then we effectively end up with 100 independent needles in a high-dimensional haystack that we need to locate and evaluate. It is our opinion that maximizing this type of function is likely hopeless and that it is probably not reflective of practical problems anyway.

4.2 Using Derivatives: Connections with Local Search

Local search algorithms have achieved a high degree of theoretical and practical success and sophistication during their development [5]. Driven by advances in computer hardware and the desire to solve increasingly interesting and difficult problems, a suite of methods has been developed that were all based on the same principle: successive local polynomial approximations. In a sense, these are also "response surface" methods as defined earlier; they construct a surrogate function, in this case a low-order polynomial, and use this model to decide where to evaluate the function next. A new polynomial approximation is then constructed using this new point. In effect, all optimization techniques for smooth functions boil down to this approach since low-order polynomials are the only functions whose optima can be expressed in a convenient algebraic form. Fortunately, all sufficiently differentiable

functions are locally polynomial, so these provide an especially good approximation over sufficiently short distances as promised by Taylor's theorem [34]. The advantage of this property is clear when we examine the performance of these techniques. Newton's method for optimization, which uses a local quadratic approximation, has a super-linear convergence rate. That is, for approximations constructed sufficiently close to an optimum, each step of Newton's method results in a squaring of the current error

$$\epsilon_{i+1} \propto \epsilon_i^2 \tag{4.5}$$

where ϵ_i is the current distance to the optimum. This "superlinear convergence" where the error at the next step is proportional to the current error raised to some power $\alpha > 1$ is a property of most techniques that construct a local polynomial model of the function to direct the search. Brent [7] shows that any algorithm based on "successive interpolation"—i.e. alternating interpolation and optimization—using polynomials converges at a super-linear rate.

The fastest of the classical local search techniques, however, use information about the derivative of the objective to construct these polynomial models, either explicitly by evaluating the gradient $\nabla_f(x)$ and possibly the Hessian $H_f(x)$ of f at the current point x, or by approximating these quantities. Newton's method for minimization requires $\nabla_f(x)$ and $H_f(x)$ explicitly, but there are several "quasi-Newton" methods that approximate one or both of these quantities. For example, the widely-used BFGS algorithm uses $\nabla_f(x)$ explicitly but iteratively constructs an approximation to $H_f(x)$. The strategy of BFGS and related methods is to forego the direct search for an optimum of f, and instead look for a point x where

$$\nabla_f(x) = 0 \tag{4.6}$$

This is a *critical point* of f, which may or may not be an optimum but in practice often is. This approach—which only computes with and reasons about derivatives—has yielded the fastest and most widely used local optimizers known, and has provided the framework for describing the rate of convergence of such methods.

This type of convergence rate analysis is common in the field of local optimization, but unheard of in the field of global optimization. From the typical global optimization perspective, even proving that a method will converge to an optimum in a finite number of steps is a difficult proposition. Attempts to narrow this performance gap have resulted in the development of the "random-restart" and other related methods mentioned earlier:

leveraging the power of local search techniques for functions that are differentiable but multi-modal can be an effective strategy, if it is possible to discover the relevant basins of attraction.

Currently, all Gaussian process based methods reason only about the function's value at points in the domain—not about the derivative of the function. However, given the kernels we have chosen, reasoning about the derivative(s) of the function at any point is equally straightforward.

4.2.1 Incorporating Derivative Information

In short, since derivative operators are linear, and since linear functions of Gaussian random variables are also Gaussian, the derivatives of sampled functions are also Gaussian random variables, provided appropriate limits exist [36]. Furthermore, since expectation is also a linear operator, prior covariances among derivatives (and between function values and derivatives) are given by derivatives of the kernel function. For example, a priori:

$$\mathrm{Cov}\left(F_z, \left.\frac{\partial F}{\partial x_j}\right|_w\right) = \frac{\partial k(z,w)}{\partial w_j} \tag{4.7}$$

and

$$\mathrm{Cov}\left(\left.\frac{\partial F}{\partial x_i}\right|_z, \left.\frac{\partial F}{\partial x_j}\right|_w\right) = \frac{\partial^2 k(z,w)}{\partial z_i \partial w_j} \tag{4.8}$$

All covariances among all derivatives and function values the points z and w are similarly computed, mutatis mutandis.[1] The notation is slightly loose here, since for some kernels the sampled functions will not have derivatives; however, the kernels we use in this work are sufficiently smooth to permit this analysis.

The ability to compute prior covariances among function values and derivatives means that we can simply include any derivative observations we might have in our observation vector $\mathbf{f}$, and then build the kernel matrix K using the above equations whenever we require a covariance that involves a derivative random variable. For example, if we observe n function values and associated gradients of dimension d, we will construct a $[n \cdot (d+1)] \times [n \cdot (d+1)]$ kernel matrix that includes all of the covariances among the function values, between the function values and the derivative values, and among the derivative values. We can then compute a posterior distribution for the function at any point using the standard formulae introduced earlier. Furthermore, we can compute the posterior mean and variance of any

[1] If the kernel is sufficiently smooth, covariances between higher derivatives are computed similarly.

derivatives we might want in a similar manner; however, we will not use this capability in our experiments.

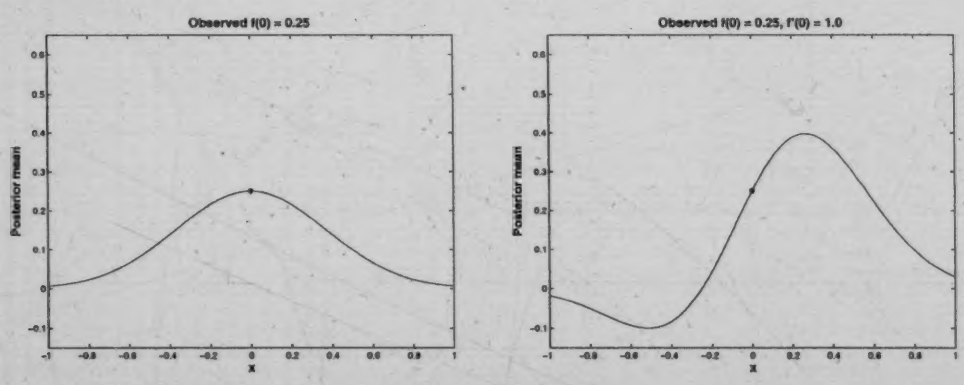


Figure 4.2: Illustration of the influence of derivative observations. On the left, one observation $f(0) = 0.25$ is used for computing the posterior mean. On the right, a function observation and a derivative observation are used: $f(0) = 0.25$ and $f'(0) = 1.0$. The change in posterior mean reflects the inclusion of derivative information. The same GP prior was used in both cases. ($\mu(x) = 0$, $\sigma_f = 1.0$, Matérn kernel with $\ell_1 \approx 0.3679$)

Including derivative information provides us with more information about the shape of the function, and hopefully allows us to identify local maxima more quickly. Figure 4.2 illustrates the effect of including a derivative observation in a 1D example, where the influence on the posterior mean is clearly visible. Note that in the example on the right, the observation vector will be:

$$\mathbf{f} = (0.25, 1.0)^\intercal \tag{4.9}$$

and the kernel matrix constructed from $k(z, w)$ will be

$$\mathrm{K} = \begin{bmatrix} k(0,0) & \frac{\partial k}{\partial z}\big|_{0,0} \\ \frac{\partial k}{\partial w}\big|_{0,0} & \frac{\partial^2 k}{\partial z \partial w}\big|_{0,0} \end{bmatrix} \tag{4.10}$$

The importance of derivative information has been shown time and again in the development of local optimization techniques, and its incorporation in Gaussian process models is straightforward. However, the impact of derivative information on Bayesian optimization has not previously been studied; therefore, we will investigate the performance Bayesian optimization using the MEI_r and MPI_r objectives when first derivative information is available.

Chapter 5

Empirical Results

In this study, our intent is to learn about the impact of four important factors: the effect of ξ_r, the effect of using MAP instead of ML for learning length scales, the effect of mismatched priors, and the effect of providing gradient information.

5.1 Experimental Setup

Each experiment we run is based on 500 functions sampled over $[-1, 1]^d$ from a Gaussian process with a constant prior mean $\mu_0(x) = 0$, a signal variance $\sigma_f^2 = 1$, and a noise variance $\sigma_n^2 = e^{-10}$. The kernels used are either squared exponential or Matérn, and are axis-scaled isotropic with a length scale for each dimension. Each function is sampled by choosing 100 uniformly randomly drawn points in $[-1, 1]^d$, sampling function values for these points, and then treating them as observed values of an underlying function. The posterior mean given these function values is the test function. All experiments that have the same kernel and length scales use the same set of 500 test functions. (This is ensured by using the same random seed for all experiments.)

For each sampled function, we run each algorithm for 30 acquisition steps, starting from the origin[1]. GP-based methods that take a parameter ξ_r have this fixed for the duration of the experiment. For each experiment, we report box plots in Appendix A that show the distribution of performance over these 500 trials for each step from 1 to 30. We use 6 different test kernels, which generate a total of 30 000 test functions. Each of these is optimized using the MEI_r and MPI_r criteria at 5 different levels of ξ_r each, and using at least 4 different priors on kernel parameters, resulting in more than 1.2 million optimization runs.

[1] Before seeing data, none of these algorithms has a prior preference on where to acquire points. Therefore the first point of each run is at the origin in order to reduce variance across methods.

Performance is measured by absolute error, assuming each algorithm reports the best function value it has observed so far. The true maximum of the test function is estimated by hill-climbing using Newton's method (i.e. with the true Hessian of the test function) starting from the location of the best of the test function's 100 observed function values. The box plots have hollow boxes but filled "notches" which are of the width suggested by McGill, Tukey, and Larsen [30] of a 95% confidence interval around the median. This interval is 1.57 times the interquartile range, divided by the square root of the number of observations, centered about the median.

5.1.1 Test Model Choices

We use test models with different dimensionality, kernel choice, and length scale in our experiments:

Dimensions	Kernel	Length Scales
2	Squared Exponential	-1.9836, -1.9836
2	Squared Exponential	-3.0000, -0.9018
2	Matérn	-1.4343, -1.4343
2	Matérn	-2.4507, -0.3525
8	Squared Exponential	-0.7629, -0.7629, -0.7629, 3.0000, 3.0000 ...
32	Squared Exponential	-0.5593, -0.5593, -0.5593, 4.0000, 4.0000 ...

With this set of models, we have tried to cover several different scenarios, some where all dimensions are equally important some where they differ, some with the smooth squared exponential kernel and some with the rougher Matérn kernel. All of the models have an expected Euler characteristic of 0.5, in an attempt to keep their complexity approximately the same even thought their length scales, kernel, and dimension may differ.

We have only two higher-dimensional mainly because of computational concerns; although posterior inference in a Gaussian process is frequently touted as taking only $\mathcal{O}(n^2)$ time once we have factored the Kernel matrix, parameter learning requires that we build the kernel matrix, factor it, and construct its $d+1$ partial derivatives at each step, which takes $\mathcal{O}(n^3 + (d+1)n^2)$ time. This coupled with the need for the hill-climber to keep track of an increasing number of dimensions means that, especially when we are optimizing thousands of function, things can take a long time.

5.2 Discussion

We now examine and interpret our empirical results and what they have to say about the exploration parameter ξ_r, about the use of priors for parameter learning, about the

effect of having the wrong model, and about the utility of gradient observations in global optimization. We summarize our results in a way that will hopefully allow others to use these techniques to solve their own optimization problems.

5.2.1 The Effect of ξ_r

To date, none of the algorithms we have examined have a principled way of managing how "local" or "global" their search procedure is, though they all have some sort of "exploration parameter," whether in the model or the acquisition criterion, that is supposed to control this aspect of the algorithm. The real reason for this is that the acquisition criteria can only approximate the full Bayes optimal procedure for choosing points to evaluate, and it is presumed that any one-step algorithm will be too "greedy" initially, and will need to be forced to "explore" more, especially when there are only few function evaluations. When using small numbers of test functions, scheduling the exploration parameter has been to be useful in some cases [41, 39].

First, we examine the case where we do not do any parameter learning, but fix all the kernel parameters. Figures 5.1 and 5.2 show the first three, middle three, and last three acquisitions from five different experiments where the optimization model's length scales are *not* learned, but are fixed to the correct values—the values that match the test GP. The performance of independently acquiring points using a Latin hypercube is shown for comparison. There is a striking uniformity among all of these results. Most notably, in all cases, using MEI_r with $\xi_r = 0.01$ is never significantly worse than the choice that gives the best median performance. This runs completely counter to conventional wisdom, which would claim that larger values of ξ_r would perform better early on and smaller values would perform better later.

This phenomenon is observed, however, when using the MPI_r criterion. The optimal value of ξ_r seems to vary between 0.1 and 1.0 depending on how many acquisitions are made, though after 30 acquisitions $\xi_r = 0.01$ was best in all cases.

Figure 5.3 shows two cases where the prior is fixed and incorrect. In these cases, the test kernel is squared exponential and the optimization kernel is Matérn, or vice-versa. Even in these cases where the model is wrong we see the same behaviour of MEI_r and MPI_r as before.

All of this evidence runs contrary to the belief that it is important to control exploration depending on how many acquisitions are to be made. It is certainly not proof that scheduling

ξ_r would not improve performance, particularly for MPI_r where there seems to be a slight need to explore more early on; nevertheless, it does not seem to suggest that there are significant gains to be made by scheduling, at least on average. For the remainder of this work, we will use MPI_r at $\xi_r = 0.01$ and MPI_r at $\xi_r = 0.1$ as points of comparison, since these settings appear to be optimal or nearly so in all of our experiments[2]. Furthermore, their performance is very similar; it would seem that in most cases, they are "safe" choices even when we must learn the kernel parameters, as demonstrate in the next section.

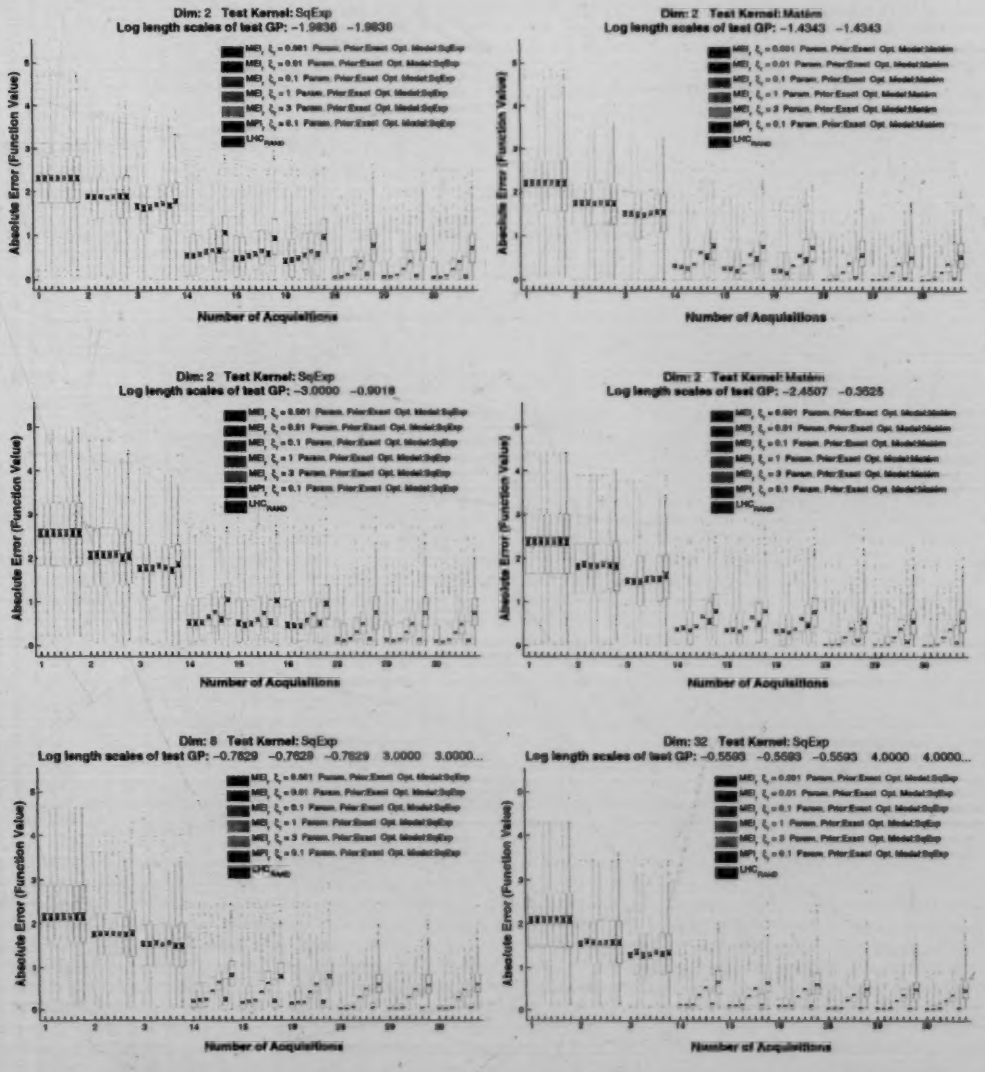

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.5625
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.1: Effect of ξ_r on the performance of MEI when priors are correct and fixed. In all cases, using $\xi_r = 0.01$ is never significantly worse than the choice that gives the best median performance.

[2]Excepting those where we also observe ∇f; see Section 5.2.4.

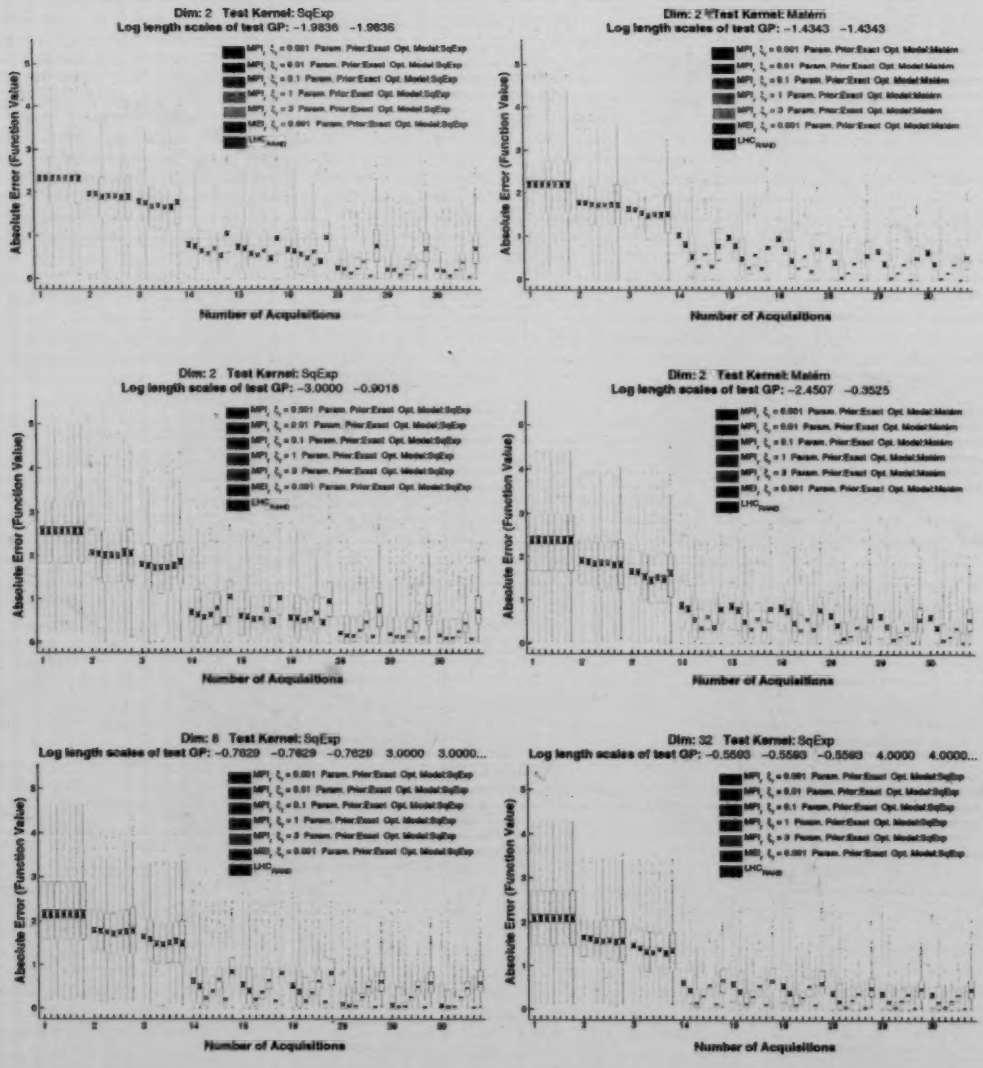

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.2: Effect of ξ_r on the performance of MPI when priors are correct and fixed. In all cases, after 30 acquisitions, using $\xi_r = 0.1$ provides the best performance. In some models, however, performance is improved by using larger ξ_r for smaller numbers of function evaluations.

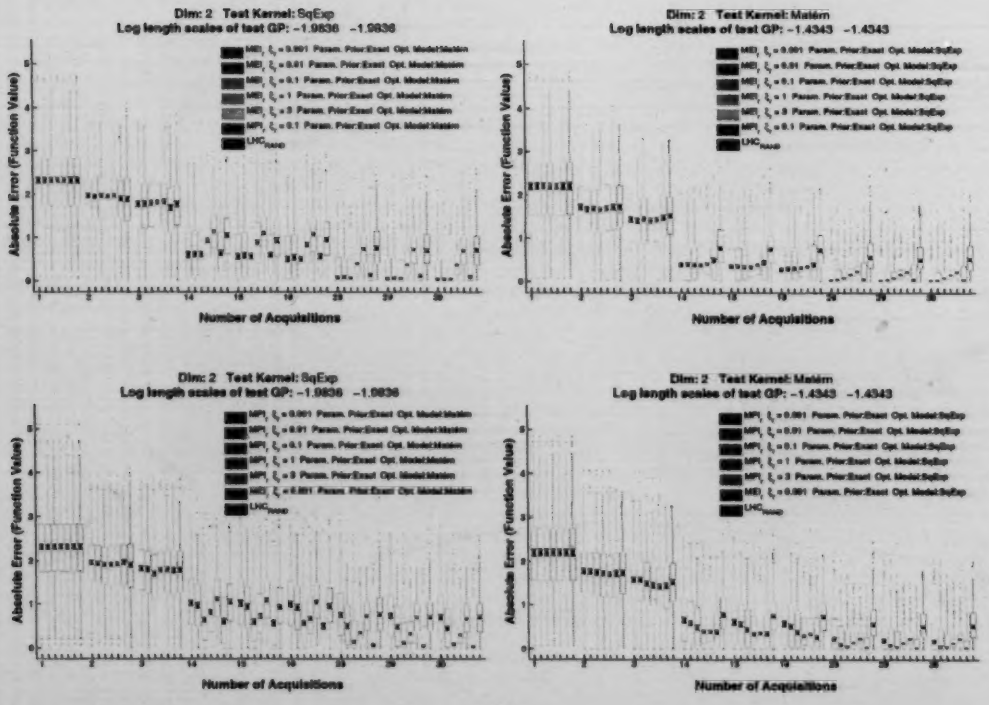

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.3: Effect of ξ_τ when the priors are incorrect and fixed. In these experiments, the test kernels have different parametric forms and fixed length scales than the optimization model kernels, though they both have EEC = 0.5. Behaviour is similar to when priors are correct and fixed.

5.2.2 The Effect of Priors on Performance

Next we investigate the case where we learn the kernel parameters as we accumulate data. Remarkably, the effect of changing ξ_r is basically the same when learning length scales as when they are fixed; again there seems to be little benefit to choosing a fixed ξ_r to suit a particular number of acquisitions. Graphs comparing ξ_r versus MAP learning with different priors are given in Appendix B. As in the exact prior case, we choose $\xi_r = 0.001$ for MEI_r and $\xi_r = 0.1$ for MPI_r in our comparisons. A summary of these runs is shown in Figure 5.4.

In all cases, there is a performance loss associated with having to learn the kernel parameters. However, as the number of acquisitions increases, performance improves to near that achieved by using the exact priors. Also, in most cases, performance earlier on is no worse than acquisition on a random Latin hypercube. This is significant because in previous work Latin hypercube acquisition is always used for the first $10 \cdot d$ acquired points *before* acquiring using MPI or MEI begins. Our results show that even using maximum likelihood (i.e. with no prior on parameters) from the very beginning we can achieve significantly better performance in far fewer function evaluations.

That said, there are cases where maximum likelihood learning can perform worse than random, as illustrated in the first graph of Figure 5.4. We hypothesize that this is because the first test model has the shortest length scales, and we have found that maximum likelihood learning tends to generate very long length scales, as was shown in Chapter 3. Introducing an ILN or EEC prior on length scales corrects this problem.

In most cases, performance when using MAP learning with ILN priors is not significantly worse than using no priors at all, and is sometimes significantly better. On the two-dimensional test model with one very short length scale of $e^{-3.0}$, using ILN priors with MEI_r is just barely significantly worse than with no priors at all. Performance using the EEC prior is similar to that using ILN, being slightly worse or slightly better depending on the situation. In any event, using a prior, whether ILN or EEC, when learning length scales seems to be the "safe" choice in most cases.

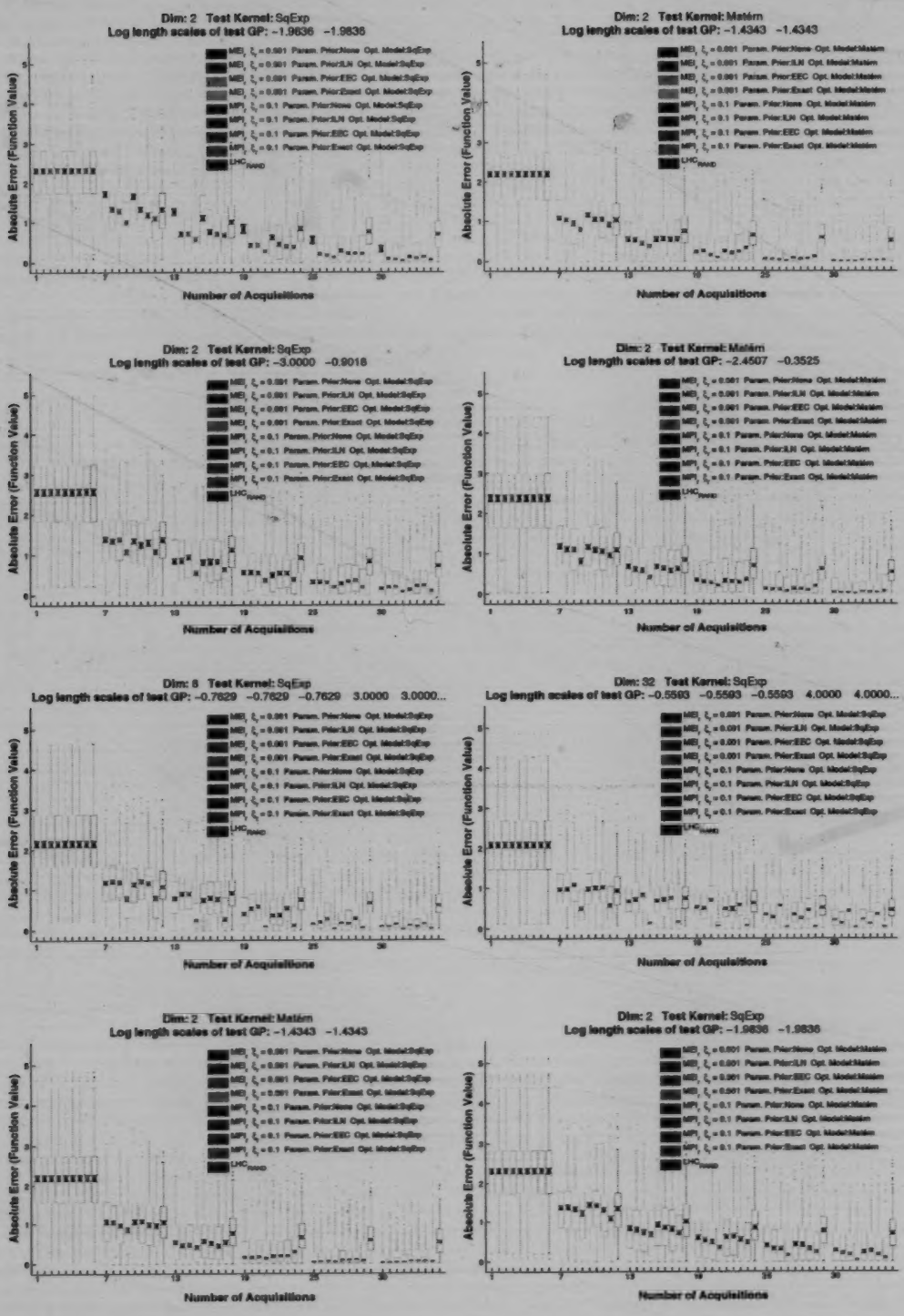

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.4: Performance of MEI_τ and MPI_τ using MAP learning with different parameter priors

5.2.3 Effect of Mismatched Models

Figure 5.5 shows the impact of trying to learn in the case where the true kernel of the test function cannot be expressed in the parametric form of the optimization model kernel. In machine learning parlance, we might say that this is an "unrealizable" case, or that the true kernel is not in the version space of kernels we are considering. Of course, our objective is not to learn the kernel of the test function but to optimize it, so the test model kernel's presence in the version space may not matter, especially if there is a kernel in our version space that behaves sufficiently like the true kernel.

Our experiments have indicated that for both MEI_r and MPI_r, when *not* learning length scales, the matching kernel performs at least as well as the the non-matching kernel, and sometimes significantly better. This is true also when using a MAP objective with either ILN or EEC priors, although in the case where the test function was Matérn and learning was done via MAP, the choice of optimization kernel did not appear to matter.

When using no prior on parameters and using maximum likelihood, however, the Matérn kernel performed as well or better than the squared exponential kernel, and seemed to suffer less from the loss in initial performance described above.

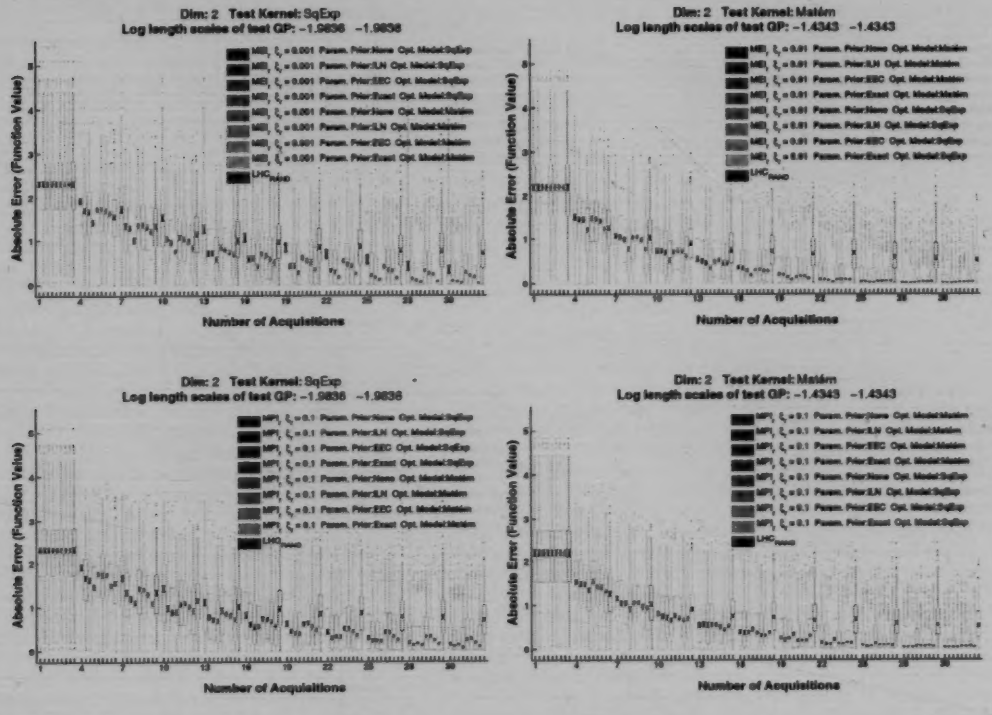

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.5: Effect of mismatched kernels

5.2.4 The Effect of Gradient Information

All the experiments discussed so far, along with all previous work in GP-based optimization, assume that we can only observe the value of $f(x)$ at the points we choose. However, as we have said, the underlying Gaussian process models we use can condition on gradient information when such information is available, because the gradient observations are jointly Gaussian with the function evaluations and with each other. Therefore, we can use all of the same techniques for learning kernel parameters and for choosing acquisition points that we used previously, augment the model with gradient information, and explore the effects of different ξ_r and different parameter priors.

Detailed results from two of the test kernels are presented in Appendix B. As we found in our function-value-only experiments, it appears that the best fixed ξ_r for the MEI_r criterion does not vary significantly regardless of the number of acquisitions; we use a fixed value of $\xi_r = 0.001$ for all of our comparisons, which is smaller than the value we chose for our function-value-only experiments.

Unfortunately, the MPI_r criterion seems to have more variable performance depending on the test kernel and the method of learning parameters. The best choice of ξ_r ranges from 0.01 to 1.0, although differences in performance seem not to be very drastic in many scenarios. We chose $\xi_r = 0.1$ as our point of comparison because it performed at or near the best when learning kernel parameters, whether using ML or MAP. However, to achieve similar performance using exact priors, a setting of $\xi_r = 1.0$ is required.

Despite the apparent sensitivity of MPI_r, it appears that MEI_r performs well at $\xi_r = 0.001$ regardless of test model or parameters, and when learning parameters performs slightly better with an EEC prior than with an ILN prior or none at all.

As another point of comparison, we include data on the performance of using a hill-climber with random restarts, which is an optimization strategy that is very common in practice. The hill-climbing data are generated as follows: Starting from the origin, the BFGS hill-climbing algorithm is applied but the number of function evaluations allowed is limited to 30. If the hill-climber converges before its alloted function evaluations are expended, it is restarted at a uniformly randomly drawn point in the domain. This continues until 30 function evaluations have been made. The reported value for BFGS after k acquisitions is the best observed value up to that point. Performance of this approach is denoted $\text{BFGS}_{\text{RESTARTS}}$ and is shown in green box plots.

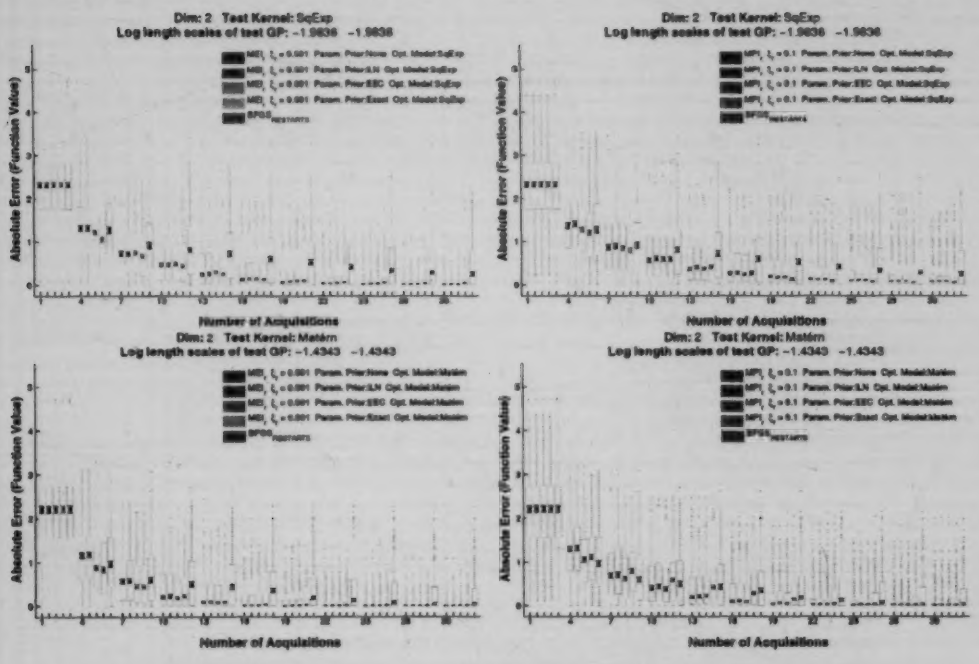

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Absolute Error (Function Value)
Number of Acquisitions

Figure 5.6: Performance of MEI_τ and MPI_τ with gradients

The performance of MEI_τ on test functions is ahead of BFGS after only a few function evaluations, and is soon very far ahead: After 15 function evaluations in both test cases of 500 functions each, 75% of the values found by MEI_τ are better than the median performance of $\text{BFGS}_{\text{RESTARTS}}$. This is a stunning improvement over a method that is widely considered state-of-the-art, albeit at a significantly higher computational overhead—recall that our method actually calls BFGS as a subroutine. Nonetheless, for expensive functions expending the extra effort in order to make fewer function evaluations may be an attractive tradeoff.

We have restricted our experiments with gradients to two-dimensional test kernels only. This is because the size of the kernel matrix increases much more rapidly for high-dimensional problems with gradient information: Each time we acquire a point, we effectively add $(d+1)$ data values to our model, and the size of K increases accordingly. This means that, for higher dimensional problems, clever methods of increasing computational efficiency like those discussed in Section 7.2.1 will become a necessity. For now, we present these results as a very encouraging first step toward using GP-based optimization with gradients.

5.3 Summary

We present a summary of the lessons we have learned from this study, directed toward anyone interested in using these techniques on their own optimization problems.

- An initial Latin hypercube acquisition is unnecessary; performance exceeding that of LHC acquisition is achieved in far fewer than the $10 \cdot d$ function evaluations typically allocated to this technique.
- When optimizing using function evaluations only, performance for MEI_τ is good at around $\xi_\tau = 0.01$, and performance of MPI_τ is good at around $\xi_\tau = 0.1$. Of course, our empirical evaluation is not exhaustive (nor can it be) but these values have worked well on average for the 30 000 test functions we used.
- When learning kernel parameters, using an ILN or EEC prior is helpful in some cases, and not damaging in any. The ILN prior in particular is easy to implement.
- When using gradient information, MEI_τ is preferable since it is more robust to different ξ_τ and models. A small $\xi_\tau = 0.001$ is effective in the cases we examined.
- Using an EEC prior when learning kernel parameters using gradient information may result in somewhat better performance, though the technique does not appear to require it.

5.4 Functions from the Literature

We now examine five analytic functions used in recent dissertations on Gaussian process-based optimization. Note that in the literature functions are typically **minimized**, and for this section only we will assume our intent is to minimize the provided functions. (Of course all of our techniques apply simply by negating function values and gradients.) The set of functions presented here consists of the union of the examples described by Schonlau [41] and Sasena [39] that existed prior to these works in the literature[3]. These functions are prominent examples of the most commonly used evaluation technique in global optimization prior to this dissertation: They have been selected because they were deemed to have characteristics that are somehow representative of the functions likely to be encountered in practice. While we believe that this approach is not as rigorous or informative as the

[3]Sasena also includes a 'mystery function,' whose "origins are unknown." [39]. We omit this function.

study we have described in the previous sections, we present a brief survey of some of these functions along with some results. The functions we use are as follows:

Branin This two-dimensional function is the sum of a quartic polynomial and a sinusoid of the first variable.

$$\begin{aligned} f(x_1,x_2) &= \left(x_2-(5.1/4\pi^2)x_1^2+(5/\pi)x_1-6\right)^2+10(1-1/8\pi)\cos(x_1)+10 \\ x_1 &\in [-5,10] \\ x_2 &\in [0,15] \end{aligned}$$

Three identical global minima with value ≈ 0.3979 are reported by Sasena [39] at $\mathbf{x}=(3.1416,2.2750)$, $\mathbf{x}=(9.4248,2.4750)$ and $\mathbf{x}=(-3.1416,12.2750)$ [39].

Goldstein-Price This function is an eighth-degree polynomial of two variables.

$$\begin{aligned} f(x_1,x_2) &= \left[1+(x_1+x_2+1)^2\cdot(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right] \\ &\quad \times\left[30+(2x_1-3x_2)^2\cdot(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right] \\ x_1,x_2 &\in [-2,2] \end{aligned}$$

Schonlau [41] reports a global minimum equal to 3.0 at $(0,-1)$.

Hartman 6 This is a six-dimensional function of the form

$$f(x_1,\ldots,x_6)=-\sum_{i=1}^{4}\exp\left[-\sum_{j=1}^{6}\alpha_{ij}(x_j-p_{ij})^2\right]$$

The constants α_{ij}, c_i and p_{ij} are given by:

$$\begin{aligned} [\alpha_{ij}] &= \begin{bmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ .05 & 10 & 17 & .1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & .06 & 10 & .1 & 14 \end{bmatrix} \\ [c_i] &= (1,1.2,3,3.2)^{\mathsf{T}} \\ [p_{ij}] &= \begin{bmatrix} .1312 & .1696 & .5569 & .0124 & .8283 & .5886 \\ .2329 & .4135 & .8307 & .3736 & .1004 & .9991 \\ .2348 & .1451 & .3522 & .2883 & .3047 & .6650 \\ .4047 & .8828 & .8732 & .5743 & .1091 & .0381 \end{bmatrix} \end{aligned}$$

Schonlau [41] reports a single global minimum of -0.30098 (-1.20069 on a transformed negative-log scale) but does not give its location.

Shekel 10 This four-dimensional function is given by

$$\begin{aligned} f(x_1,\ldots,x_4) &= -\sum_{i=1}^{10}\frac{1}{||\mathbf{x}-\mathbf{a}^i||^2+c_i} \\ x_1,x_2,x_3,x_4 &\in [0,10] \end{aligned}$$

where $\mathbf{x} = (x_1, x_2, x_3, x_4)$. The vectors $\mathbf{a}^i$ and constants c_i are given by

$$
\begin{aligned}
[\mathbf{a}_i] &= \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix} \\
[c_i] &= (.1, .2, .2, .4, .4, .6, .3, .7, .5, .5)^{\mathsf{T}}
\end{aligned}
$$

Schonlau [41] reports a global optimum near (but not at) $\mathbf{x} = (4, 4, 4, 4)$. Its value is not given but appears to be approximately -10.52.

Six Hump Camelback This function is a quartic polynomial of two variables.

$$
\begin{aligned}
f(x_1, x_2) &= (4 - 2.1x_1^2 + x_2^3) \cdot x_1^2 + x_1 x_2 + (-4 + 4x_2^2) \cdot x_2^2 \\
x_1 &\in [-2, 2] \\
x_2 &\in [-1, 1]
\end{aligned}
$$

Sasena [39] reports two identical global minima of -1.0316 at the domain points $(-0.0898, 0.7127)$ and $(0.0898, -0.7127)$.

A direct comparison with previous work on these test functions is difficult for at least two reasons. First, a variety of evaluation metrics have been used in the past. For example, Schonlau assumes knowledge of the true global minimum, and counts the number of function evaluations necessary to achieve an absolute tolerance of 0.0001 on the function value. Sasena on the other hand counts the number of function evaluations to reach within 1% of the value of the global minimum, along with several other measures. Second, the method of acquiring results varies from study to study. Schonlau reports results of one attempt at optimization, while Sasena reports the average results from 10 different sets of initial starting acquisitions. The results we present here are similar to those presented earlier: we simply allow the algorithms 30 function evaluations, and compare against uniformly randomly sampled points (when no gradient is used) and BFGS with random restarts (when the gradient is made available.)

Figures 5.7, 5.8, 5.9, and 5.10 each illustrate one optimization run on each of the above functions using all combinations of kernel, criterion (MPI_τ and MEI_τ), and length scale

prior. Figures 5.7 and 5.8 illustrate performance without gradient observations and Figures 5.9, and 5.10 illustrate performance with gradient observations. Previously reported global minima are indicated by a black horizontal line.

The first thing to note here is that these runs present at best a sanity check; no reasonable general inferences can be drawn from such a small set of observations. Nevertheless, some interesting points arise. First, it appears that the Branin function is particularly easy to optimize; all methods including random sampling find a very good point after very few iterations. We conjecture that this is because the function has three identical global minima, each of which lies in a wide, shallow basin. On the other hand, the unique minimum of the Goldstein-Price function lies in a small convex basin surrounded by concavities, and proves to be more difficult. Indeed, it appears that providing gradient information about the Goldstein-Price function is unhelpful, possibly because it is not well-modeled by the kernels we chose, or because the concavities near the minimum are misleading to the search. The Hartman 6 and Shekel 10 functions are both constructed by summing up several 'wells' that each contain a local minimum. Those of the Shekel 10 are much narrower and deeper than those of the Hartman 6. On these functions the GP models made better progress, and gradient information was beneficial in both cases. In particular, it appears that a global model of the Shekel 10 function is beneficial, perhaps because it allows a systematic search of the local minima in order to find the global minimum; BFGS with random restarts did not manage to locate the deepest well, and simple random samples did not find any of them. Finally, the Six Hump Camelback function presents an interesting case: As mentioned above, Sasena reports that the global minimum of the function is -1.0316, but all of our methods found values well below this. This is because, over the domain indicated, the global minimum is against a constraint: Because the dominant x_1 term is $-2.1x_1^4$, eventually the function will tend to $-\infty$ as we increase $|x_1|$. In fact, it falls off sharply just before the edges of the domain, resulting in several global minima against active constraints. It is not clear why this was not discovered or mentioned previously, but it does illustrate the importance of properly handling this type of optimum.

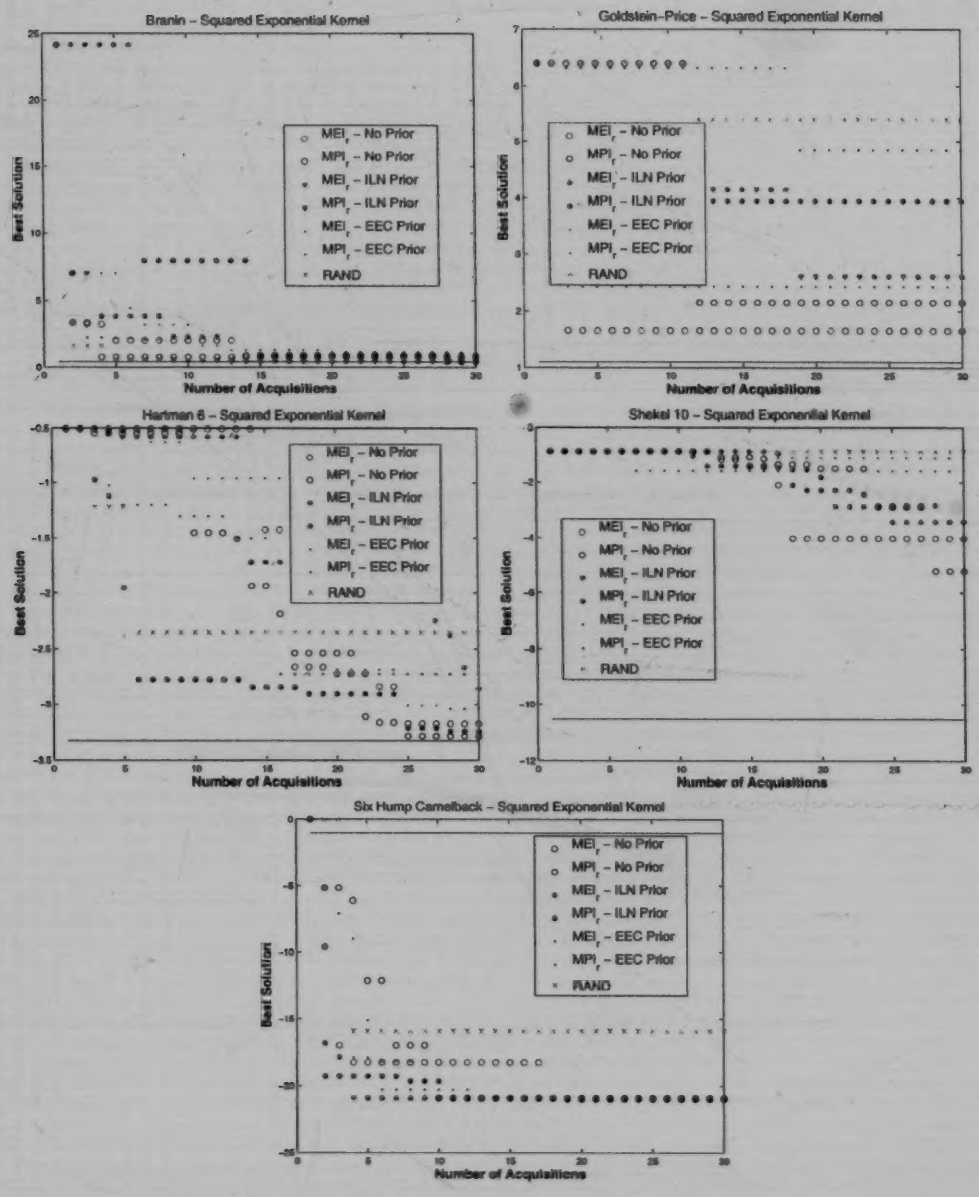

Branin – Squared Exponential Kernel
Goldstein–Price – Squared Exponential Kernel
Hartman 6 – Squared Exponential Kernel
Shekel 10 – Squared Exponential Kernel
Six Hump Camelback – Squared Exponential Kernel
Best Solution
Number of Acquisitions
MEI_r – No Prior
MPI_r – No Prior
MEI_r – ILN Prior
MPI_r – ILN Prior
MEI_r – EEC Prior
MPI_r – EEC Prior
RAND

Figure 5.7: Analytic function examples using the squared exponential kernel. For MPI, $\xi_r = 0.1$. For MEI, $\xi_r = 0.01$. No gradient information is provided. The solid black line indicates the best previously published minimum.

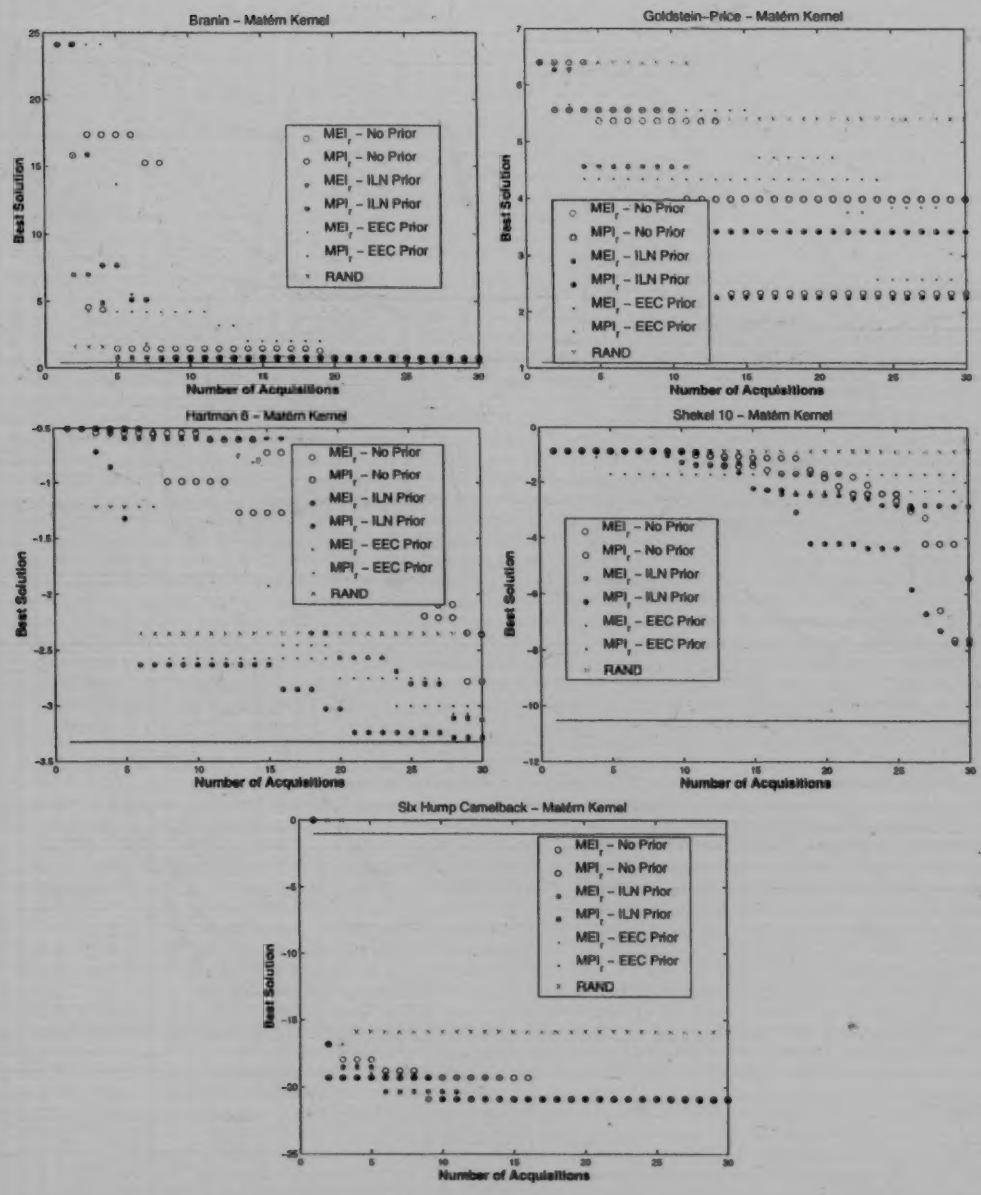


Figure 5.8: Analytic function examples using the Matérn kernel. For MPI, $\xi_r = 0.1$. For MEI, $\xi_r = 0.01$. No gradient information is provided. The solid black line indicates the best previously published minimum.

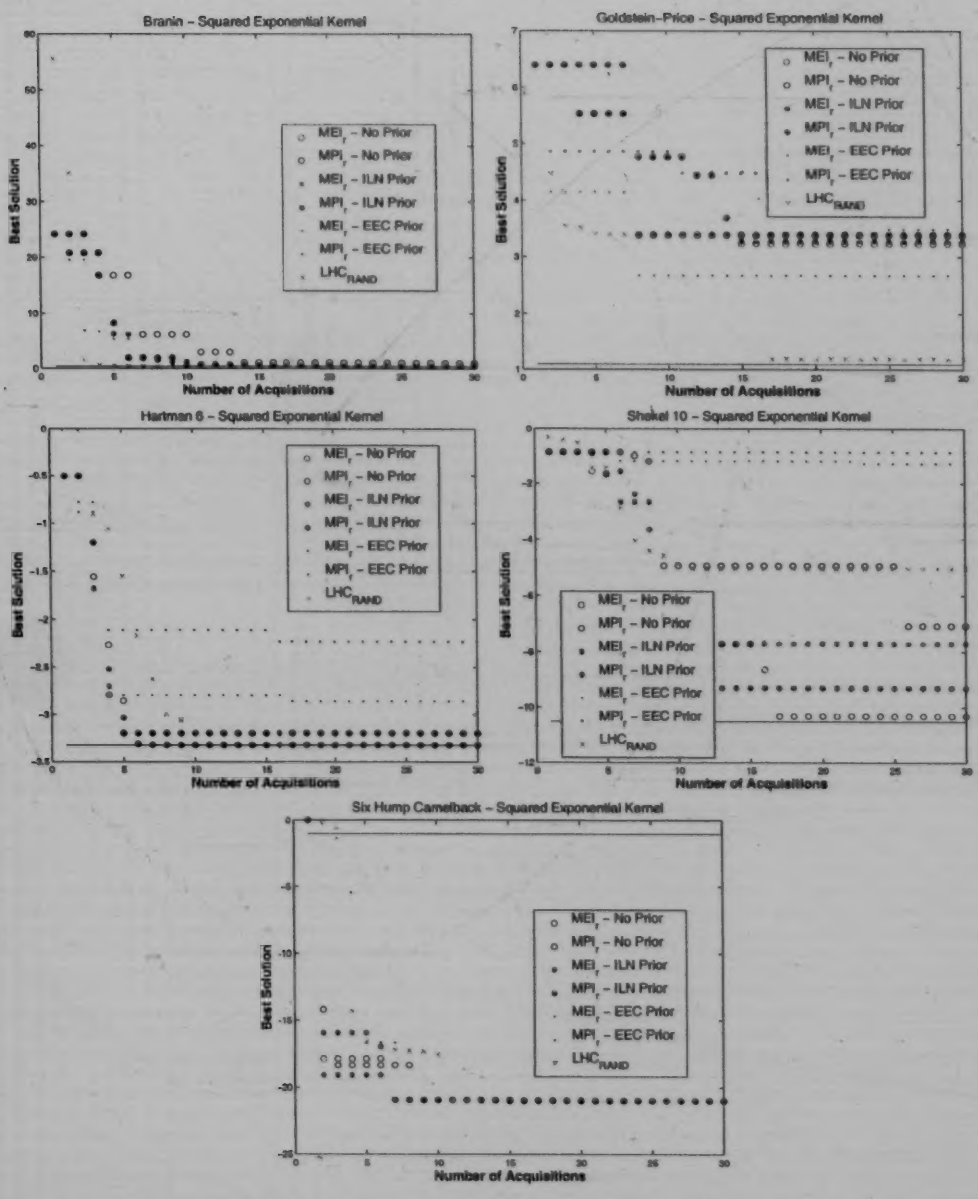

Branin – Squared Exponential Kernel
Goldstein–Price – Squared Exponential Kernel
Hartman 6 – Squared Exponential Kernel
Shekel 10 – Squared Exponential Kernel
Six Hump Camelback – Squared Exponential Kernel
Best Solution
Number of Acquisitions
MEI$_r$ – No Prior
MPI$_r$ – No Prior
MEI$_r$ – ILN Prior
MPI$_r$ – ILN Prior
MEI$_r$ – EEC Prior
MPI$_r$ – EEC Prior
LHC$_{RAND}$

Figure 5.9: Analytic function examples using the squared exponential kernel. For MPI, $\xi_r = 0.1$. For MEI, $\xi_r = 0.001$. Gradient information is provided. The solid black line indicates the best previously published minimum.

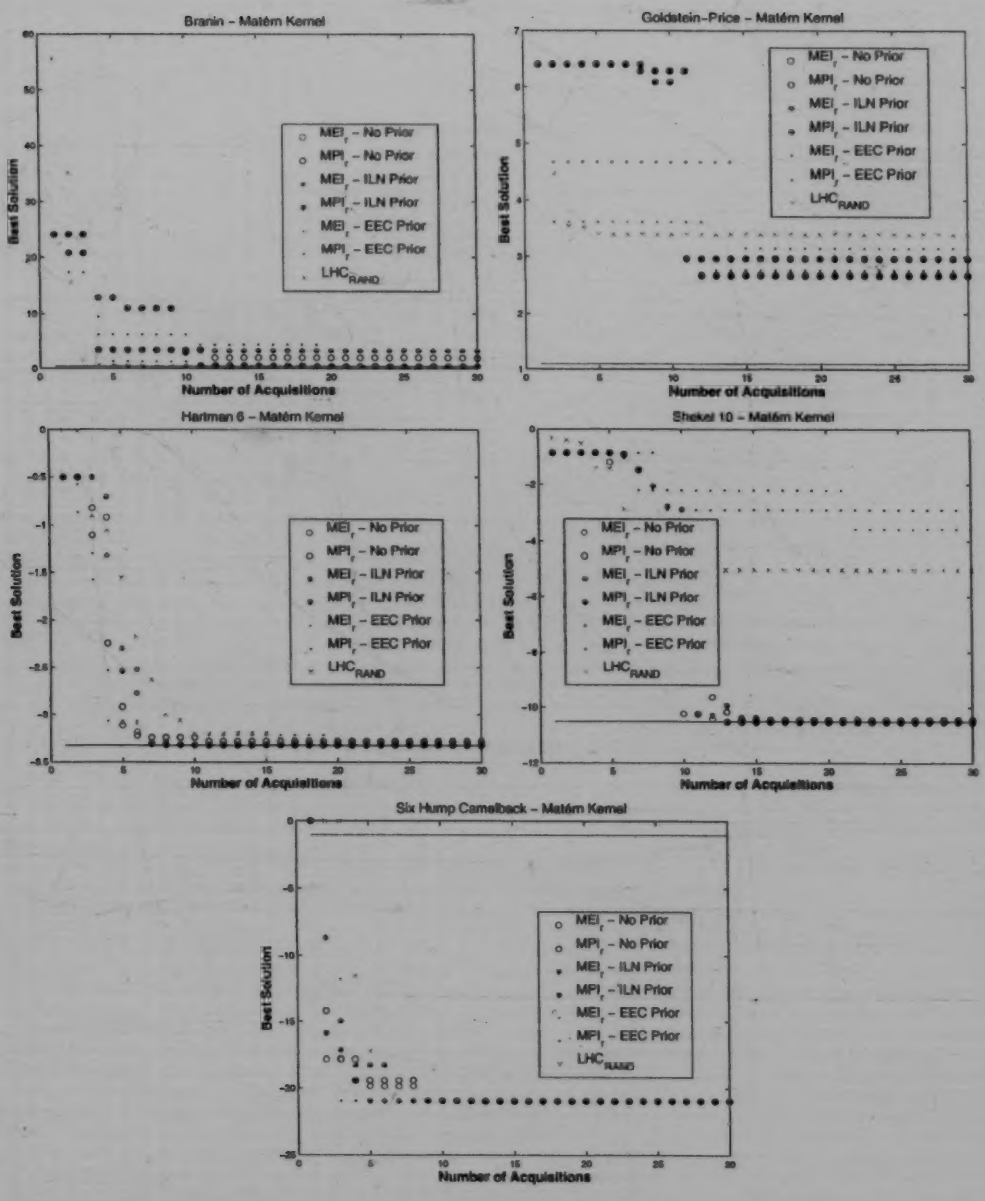


Figure 5.10: Analytic function examples using the Matérn kernel. For MPI, $\xi_r = 0.1$. For MEI, $\xi_r = 0.001$. Gradient information is provided. The solid black line indicates the best previously published minimum.

Chapter 6

Applications

To date, we have applied Gaussian process-based optimization in two domains of interest to computing scientists: Improving the speed and smoothness of robot walking gait optimization, and reducing the error of a stereo matching algorithm. The work on AIBO gait optimization was conceived early on in the course of the thesis, and is a proof of concept that illustrates how even a very simple version of Gaussian-process based optimization can be effective in higher dimensional spaces than have previously been addressed. Most recently, we have begun collaboration on a large-scale comparison of stereo matching algorithms, and we have used the knowledge gained in our empirical studies to help develop a fast and consistent optimization procedure for different stereo matching approaches.

6.1 AIBO Gait Optimization

Legged robot platforms offer many advantages over traditional wheeled robots. Their unique ability to traverse a wide variety of commonly encountered terrain makes walking robots are basically a requirement for performing useful tasks in our human-centric world. Despite these advantages, legged robots only slowly being adopted, in part because of the difficult challenge of developing an effective walking algorithm or "gait".

Optimizing a robot's gait is a complex problem. An open loop gait consists of a sequence of joint values for each of the many degrees of freedom in a legged robot. Simplified parametric representations of leg trajectories can result in a manageable number of parameters; however, these parameters can have complicated interactions making manual tuning of gait parameters time-consuming for simple robots and nearly impossible for the increasing complexity of humanoid platforms. Worse yet, no single gait will be effective in all circumstances. The walking surface is critical and can vary in terms of friction, softness,

and height variation (e.g., compare concrete, linoleum, carpet, and grass). Robot platforms themselves also vary due to manufacturing imperfections and general joint and motor wear. Lastly, even measuring the quality of a gait is likely to be situation specific. Although velocity may seem like the obvious choice of objective function, relative stability of the robot's sensor hardware can also be important, for example. Tailoring a robot's gait to the environmental, robot, and task specific circumstances may be possible to some degree, but relying on constant manual re-tuning is largely impractical. Automatic gait optimization is an attractive alternative to this laborious manual process, and has been explored in the past using hill-climbing and genetic algorithm based approaches.

We applied Bayesian optimization to the problem of gait optimization and demonstrate the effectiveness of the approach using the Sony AIBO quadruped robot [27]. We show that when applied to two different objectives the Gaussian process model not only finds effective gait parameters, but also does so with an order of magnitude fewer evaluations than a local search competitor.

Previous Approaches The Sony AIBO, a commercially available quadruped robot, has spurred recent interest in gait optimization, driven primarily by its use in the RoboCup Legged Soccer League, where well tuned walks are a requirement for success. Number of recent approaches for gait tuning have been pioneered on the AIBO.

A common foundation for all of these approaches including our own is the notion of a "walk engine", or parameterized gait generation system. Since the number of degrees of freedom in legged robots is large, optimizing the angle of each joint at a finely discretized time scale would involve searching over thousands of parameters, a task typically considered intractable. The walk engine reduces the number of parameters by focusing on leg trajectories that are both physically possible and intuitively plausible. These parameters usually define properties of each leg's trajectory such as the "distance the foot is lifted off the ground" and "the period of the walk in milliseconds". The space of parameterized walks obviously has a large impact on the final quality of any automatic gait optimization, but the optimization problem itself is the same regardless of the walk engine. Optimization in all cases requires finding a point in a parameter space with between eight and fifty dimensions that results in an effective gait. Each of the works cited below is based on a different walk engine. Because of this, and because of the variation in lab surfaces and individual robots,

reported walk speeds are largely incomparable.[1]

Despite these differences, each approach uses a similar experimental setup. All of the approaches involve evaluating specific parameter settings by having the AIBO walk in a structured arena using the parameters to be evaluated. Local sensor readings on the AIBO—either camera images of visual landmarks or the IR sensor readings of walls—are then used to compute a noisy estimate of the gait's average velocity. The procedure may be replicated several times to construct an averaged estimate with lower variance.

Genetic Algorithms An genetic algorithm inspired approach was the first proposed method for gait optimization on the AIBO. Hornby et al. [17, 16] used a standard genetic algorithm search on an early prototype AIBO. A population of parameters was maintained at each generation, a new population was formed through mutation and crossover of individuals from the previous generation, and parameters in the population that evaluated poorly were discarded. Their procedure showed slow gait improvement over 500 generations, requiring approximately twenty-five robot-hours of learning.

The evolutionary approach was revisited by Chernova and Veloso [9]. They used similar mutation and crossover operations to generate candidate gaits. Unlike the work of Hornby, their parametric space of walks included a measurement of the possibility that the AIBO could physically perform the gait. This allowed them to throw out poor gait parameters without requiring an expensive on-robot evaluation. In addition, they used a "radiation" procedure to disperse clusters of similar parameters in the population and force further search. They demonstrated that the technique learned competitive walks using 4,000 evaluations and a total running time of approximately five hours distributed across four robots for a total of twenty robot-hours.

As the AIBO's evaluations of performance are noisy, both approaches must deal with the possibility that an inaccurate evaluation will cause poor gait parameters to incorrectly remain in the population. Both used targeted reevaluation to reduce this possibility, reevaluating either the parameters that remained for multiple generations or the ones that performed disproportionately well.

Local Optimization The second family of approaches that has been explored involves adapting techniques for local function optimization to the gait optimization problem.

[1] Despite the lack of basis for comparison, the three most recent techniques discussed below all achieve a similar walk speed in the range of 0.27–0.29 m/s.

Kim and Uther [22] used Powell's method [34], which performs line search along a chosen search direction based on the effectiveness of previously chosen search directions. Kohl and Stone [23] used a hill climbing approach. Although the gradient is not known, a set of random perturbations are evaluated empirically, and these evaluations are used to construct an approximation of the gradient. The parameters are then adjusted by a fixed step size in the direction of this estimated gradient. Kohl and Stone reported the fastest learning result in the literature, requiring only three hours distributed across three robots for a total of nine robot-hours. It is important to note that the reported experiments for both techniques involved initializing the optimization with a known set of reasonable parameters. This differs from evolutionary approaches, which begin with a random initial population.

Drawbacks All of the previous approaches share three key drawbacks. First, they can get stuck at local optima. Kim and Uther reported actual experiences of local optima and Kohl and Stone noted the importance of starting from good initial parameters, having found considerably poorer performance under different starting conditions. There are techniques to deal with local optima as we have seen, such as random restarts for local function optimization approaches and radiation for evolutionary approaches. However, both involve a considerable increase in the required number of gait evaluations. Furthermore, the approaches forget previously evaluated gaits once they either die out of the population or after the gradient step is taken. Not only is this an inefficient use of expensive gait evaluations, but certain parameter settings may be unnecessarily reevaluated. Finally, none of the approaches explicitly model the noise in the evaluation process. Hence they all involve long individual evaluations or repeated shorter evaluations that are averaged to compute a less noisy estimate. The Bayesian optimization approach does not suffer these disadvantages, and consequently requires considerably fewer gait evaluations.

Gaussian Process-Based Gait Optimization Like previous approaches, we assume that we already have some parameterized walk engine with d parameters. We model the stochastic velocity function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, which maps walk parameters to velocity, as a Gaussian process. The prior mean and kernel function of the Gaussian process are chosen based on prior domain knowledge. We use the MPI acquisition criterion to decide which parameters to evaluate on the robot based on the previous observations, and after some number of gait evaluations the parameters that generated the fastest observed walk are

returned.

Walk Engine As discussed earlier, a complete joint trajectory for a gait could have thousands of parameters. All gait optimization techniques parameterize this walk space using a walk engine. For this work, we have used Carnegie Mellon University's Tekkotsu[2] software to control the AIBO, which includes a walk parameterization (as of release 2.4.1) originating from the Carnegie Mellon CMPack 2002 robot soccer team. This is an early version of the CMWalk engine used in the work by Chernova et al. [9]. In consultation with a domain expert, we identified 15 walk parameters along with reasonable bounds on each parameter to define our domain, i.e., $\mathcal{X} \subset \mathbb{R}^{15}$.

Model For this work, we will use a fixed optimization model, i.e. we will not do any parameter learning. Since we have some intuition about the behaviour of the walk function, taking such a "black-box" approach less necessary. We therefore define a prior mean function $\mu_0 : \mathcal{X} \to \mathbb{R}$ and kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We also need to specify the signal variance σ_n^2 which represents the amount of overall variation in the function, and the variance σ_n^2 of the noise that is believed to be added to each observation observation F_x. For the mean, we use a constant function $\mu(x) = \mu_f$. For the covariance function, it is hypothesized that, in general, parameter vectors that are close in terms of Euclidean distance are likely to have similar walk velocities, and therefore a large positive covariance. Furthermore, we expect that some parameters may have wide-reaching consequences, and so velocities generated by parameters that are far apart should still have some small positive correlation. We therefore chose to use the squared exponential kernel with axis-aligned scaling. We now need to simply choose the constants σ_n^2, μ_0, and σ_f^2. Setting these to appropriate values depends on the feature of the gait to be optimized. We present results both for optimizing the gait's velocity and its smoothness, so we examine the choice of prior for each of these cases in turn.

Parameters for Velocity Gait velocity has been studied relatively extensively both on the AIBO and with our particular walk engine. In consulting with our domain expert, we easily solicited useful prior information. In particular, we chose $\mu_f = 0.15$ and $\sigma_f^2 = 0.066$, which correspond to the intuition, "For some random gait parameters, we expect the observed velocity to be 0.15 meters per second and within about 0.2 meters per second 99% of the time." For observation noise, our domain expert was not familiar with our

[2]http://www.tekkotsu.org

particular experimental setup. Instead, we computed the sample variance of a small number of observations of one particular setting of the walk parameters. This gave us a value of $\sigma_n^2 = 0.01$, which seemed to work well in practice. As an interesting test of the stability of our method with respect to the model parameters, we also ran the velocity optimization with a prior variance parameter of $\sigma_f^2 = 0.66$ (ten times larger).

Prior for Smoothness Since smoothness had not been evaluated before, we had no domain expert to consult. Instead, we made a small number of acquisitions (about 30) and used sample means and variances to estimate the parameters of the prior $\mu_f = -30$, $\sigma_f^2 = 100$, $\sigma_n^2 = 2.25$. As we will show in the next paragraph, even this simple uninformed method for specifying the Gaussian process model can be very effective.

Acquisition Criterion We use the maximum probability of improvement (MPI) acquisition criterion with a fixed ξ. Since the optimization model parameters are fixed, this is equivalent to choosing a $\xi_r = \xi/\sigma_f^2$.

Implementation Details In order to compute the point of most probable improvement point, we used the generic constrained hill climber in MATLAB (`fmincon`) supplied with the function and gradient of the acquisition criterion. We used as default starting points the two best parameters found so far, and 13 drawn uniformly random within the bounded domain for a total of 15 starting points. In addition, we forced the first gait evaluation selection by choosing the center point of the domain. Since the Gaussian process model starts with a uniform belief over the domain, all function points are equally good to the most expected probable rule.

Results We have applied our Gaussian process approach to two gait optimization tasks on the Sony AIBO ERS-7. We first look at the standard problem of maximizing walk velocity, and we also examine the problem of optimizing a gait's smoothness. The goal of any gait optimization technique is to find a near-optimal gait in as few evaluations as possible. Therefore, we'll want to compare techniques using this criterion.

Since previous gait learning has not involved the same walk engine, experimental setup, or robots, comparing directly with previously reported results can be somewhat problematic. For a more direct comparison, we have implemented the hill climbing method of Kohl and Stone [23] described earlier, and applied it in identical circumstances as our approach.

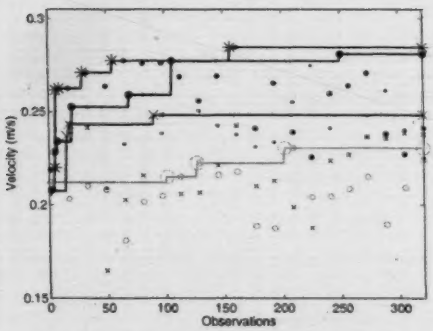
The Kohl and Stone algorithm is the most data efficient technique for the AIBO from the literature, demonstrating effective walks with only nine robot-hours of training. We replicated their experimental setup, using 15 random evaluations to approximate the gradient at each "iteration" and using a step size of 2. The empirical epsilon used in estimating the gradient was 1/15 of the parameters' range, which seemed to be an adequate change in performance. As with the Gaussian process model, we started the hill climber from the point in the center of the space.

Physical Setup To evaluate each walk parameter choice, we had the robot walk between two visual landmarks while focusing the head on the center of the landmark. The robot determined its distance from a landmark as it walked toward it based on the landmark's apparent width in the camera's field of view, and that change is used to estimate velocity. To determine a gait's smoothness, we measured the time-averaged distance from the center of the landmark to the center of the robot's field of view, and negated this. Unstable walks that result in a large amount of head movement yield negative smoothness values, since it is difficult for the robot to keep the head and the camera aimed at the target. More fluid walks allow the robot to aim the camera more directly at the target, resulting in a smoothness much nearer to zero.

Each observed measurement is the result of three "traversals" from one landmark to the other. The average time for three traversals, including time to turn around, was approximately one minute. This was chosen to be consistent with Kohl and Stone's hill climbing experiments. We could have easily only used a single traversal and compensated by increasing the observation variance used in the model.

Gait Velocity A graph showing the result of 321 observations is shown in Figure 6.1(a). We chose this number of observations a priori to allow the hill climber 20 "steps" or iterations with 15 test points for each. Both our Gaussian process technique and the Kohl and Stone hill climbing technique are shown, as well as the simple baseline of choosing gaits uniformly at random. The solid lines represent the maximum achieved walk speed over the accumulating observations, and the corresponding isolated markers show the maximum velocity achieved over the most recent 15 observations.

We found that both the Gaussian process and hill climbing methods performed appreciably better than random evaluations. The best walk velocity found was 0.285 m/s, which was



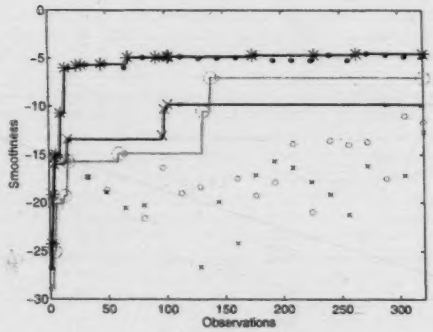
(•) GP w/MPI
0.281 m/s
$\sigma_f^2 = 0.06$

(∗) GP w/MPI
0.285 m/s
$\sigma_f^2 = 0.6$

(×) H.Clmb
0.248 m/s

(○) U.Rand
0.230 m/s

(a)



(∗) GP w/MPI
-4.53

(×) H.Clmb
-9.80

(○) U.Rand
-7.07

(b)

Figure 6.1: Results for (a) gait velocity and (b) gait smoothness. Solid lines represent cumulative maximum, and the small markers indicate the maximum observation from the last 15 observations.

found by the Gaussian process with the over-estimated prior variance, $\sigma_f^2 = 0.66$, although the walk found by the Gaussian process with the "sensibly" initialized variance is nearly as fast. Despite having already warned of the difficulties in comparing walk speeds, note that this speed is comparable to other learned gaits. More impressively, though, is the fact this speed was attained after only 120 observations, which took approximately two robot-hours. This is nearly a five-fold improvement in the required number of robot-hours.

It is interesting that the best walks found by the two Gaussian process models are in widely separated parts of the space. The walk found by the sensible setting of $\sigma_f^2 = 0.066$ has a low period and shorter stride, with a parameter vector far from the center of the space. On the other hand, the experiment at $\sigma_f^2 = 0.66$ found a similarly fast walk near the center of the space with longer, slower strides that cover a similar distance.

Although the results of the hill climbing approach were not poor, we were somewhat surprised that the performance was not better. The method tended to take very poor steps once it reached a value of about 0.230 m/s. There are two natural explanations. One may simply be local optima, which is in line with Kohl and Stone's noted importance of the initial parameter vector. Alternatively, based on the frequency of taking poor gradient steps, the step size may have been too large along certain dimensions. Random restarts and a variable step size for each dimension could mitigate these unimpressive results.

Gait Smoothness A similar graph for gait smoothness is shown in Figure 6.1(b). The Gaussian process optimization found the smoothest walk of all three methods and did so within the first 20 observations, or only twenty minutes of robot time. Later improvement was only incremental. In a post-mortem analysis, the gait smoothness task is apparently much simpler. The impact of parameters on our measure of smoothness ended up being quite simple as a wide range of walks with a short period (below about 310 ms) and a moderate requested walk speed (between 210 and 240 m/s) resulted in a smooth gait measurement. Scores of such walks were typically greater than -10. Due to the independence of this pair of parameters from the rest, it is not unlikely that even random search would find a smooth gait as choosing parameters in this range will occur on average every 250 gait evaluations. Our random search trial did happen to find one such point, giving it a moderate win over the hill climbing technique. Again, we suspect the unimpressive hill climbing result to be due to initial conditions and local maxima.

Conclusion We proposed a new approach to gait learning using Gaussian process regression for global optimization. The approach overcomes many drawbacks of previous techniques, effectively avoiding local optima, efficiently using all gait evaluations, and explicitly modelling noisy observations. Even with all of the caveats associated with comparing gait velocities and training time, we have demonstrated that the approach is not only effective in our high dimensional, noisy, non-convex, optimization problem, but also requires drastically fewer evaluations. This was also accomplished with a minimum of parameter tuning, demonstrating effective performance when using prior settings from a domain expert, incorrect settings, and even data derived settings. This work illustrates the potential for Gaussian process-based global optimization to be a useful technique for optimizing complicated functions arising in real-world problems.

6.2 Stereo Matching Parameter Optimization

Stereo matching algorithms use two or more images that have been taken from slightly different viewpoints to recover a "depth map" of the photographed scene. That is, for each pixel p_i, we would like to recover the distance from the camera to the 3D object that projects to p_i. One class of state-of-the-art techniques for this problem constructs a probabilistic graphical model of depth maps and attempts to recover the most probable map given the input images. Consider a left image $\mathbf{p}^l$, a right image $\mathbf{p}^r$, and a left depth map $\mathbf{d}^l$. One can

think of $\mathbf{p}^l$ and $\mathbf{p}^r$ as vectors of pixel intensities p_i^l, and $\mathbf{d}^l$ as a list of depth values d_i^l, one for each pixel in the left image[3]. We now assert that

$$p(\mathbf{d}^l|\mathbf{p}^r, \mathbf{p}^l) \propto p(\mathbf{p}^r|\mathbf{d}^l, \mathbf{p}^l) \cdot p(\mathbf{d}^l|\mathbf{p}^l) \tag{6.1}$$

and maximize over $\mathbf{d}^l$ to find the maximum a-posteriori (MAP) depth map. The likelihood term, $p(\mathbf{p}^r|\mathbf{d}^l, \mathbf{p}^l)$, describes how well the left image plus the left depth map are able to re-create the right image. The prior, $p(\mathbf{d}^l|\mathbf{p}^l)$, gives the probability that $\mathbf{d}^l$ is a depth map that describes the geometry in the left image.

Each of these two terms is typically expressed using a Gibbs-distributed "energy function" over images and depth maps, which are then summed to produce a total energy E such that $p(x) \propto e^{-E(x)}$. Since we are only looking for the MAP depth map and do not need its probability, we do not need to compute the log partition function and can simply solve

$$\underset{\mathbf{d}^l}{\operatorname{argmin}} \; E_{\text{data}}(\mathbf{p}^r, \mathbf{d}^l, \mathbf{p}^l) + E_{\text{smooth}}(\mathbf{d}^l, \mathbf{p}^l) \tag{6.2}$$

where E_{data} and E_{smooth} are the names typically given to the likelihood and prior energy functions, respectively. Given these functions, several techniques that exploit structure in the problem have been successfully used to compute the minimum energy (and therefore MAP) depth map, including max-flow/min-cut algorithms, dynamic programming, and branch-and-bound search.

However, the energy functions themselves are parameterized functions of the image data and depth map. For example, one simple choice for the E_{smooth} prior energy function is

$$E_{\text{smooth}}(\mathbf{d}^l, \mathbf{p}^l) = w \cdot \sum_{\{i,j\} \in \mathcal{A}} \frac{|d_i^l - d_j^l|}{|p_i^l - p_j^l|} \tag{6.3}$$

$$\mathcal{A} = \{\{i,j\} : \text{pixel } p_i^l \text{ and pixel } p_j^l \text{ are adjacent}\} \tag{6.4}$$

Pixels are typically considered "adjacent" if one is a four-neighbour of the other, i.e. they are next to each other in the up, down, left, or right direction. This function will prefer depth maps where adjacent pixels have similar depth; however if their intensity values of the pixels are different, the change in depth is allowed to be greater[4]. Note the weight w in the equation: Altering its value will alter the importance of the E_{smooth} term in our model, and will change the solution of Equation 6.2; using a larger w will produce smoother depth maps in the sense described by E_{smooth}.

[3] With a few exceptions (notably Kolmogorov's work [24]), left and right depth maps are computed independently.

[4] For this function we define $0/0 \doteq 0$ so that adjacent pixels with identical intensity and depth do not generate infinite energy.

Each term in the total energy function is given a weight, and some of the terms may have other "internal" parameters. In more complicated energy functions currently in use, there may be up to nine of these "auxiliary parameters" in the E_{smooth} and E_{data} components.

Previous Work To date, these parameters have been optimized "by hand" based on researchers' intuitions about what values are likely to perform well, and based on a small number of evaluations on datasets that have an associated ground-truth, such as laser range-finder readings. This has made comparison of different stereo matching algorithms very difficult because the procedure for optimizing the auxiliary parameters is not standardized. Exhaustive grid search over parameter space has also been applied in some cases, but of course this takes time exponential in the number of parameters, and is impractical even in low dimensional settings when each run of the underlying minimizer takes a significant amount of time.

Gaussian Process-Based Stereo Matching Optimization Our goal is to produce a system that can optimize these parameters reliably using a small number of function evaluations (i.e. runs of the matching algorithm) so that researchers can more quickly and accurately compare techniques that use different energy functions. We have constructed a system that chooses parameters, runs the stereo matching algorithm which reports a resulting error rate with respect to ground-truth, incorporates this information into the optimization model, and chooses the next parameter setting according to the acquisition criterion [31]. Because we will be testing several different matching algorithms which have different parameters, we choose a more black-box approach than that used for AIBO gait optimization: After considering the empirical evidence produced in this thesis, we choose to use a squared exponential kernel together with MAP learning using independent log-normal priors, and the MPI_r acquisition criterion with $\xi_r = 0.1$.

The two stereo matching algorithms we investigated are similar to work by Hirschmüller based on dynamic programming [14]. In both of the algorithms, E_{data} has no free parameters, and E_{smooth} has two free parameters w_1 and w_2, which we will optimize. Both of these are constrained to the range $[1, 50]$. The smoothing energy function is given by

$$E_{\text{smooth}}(\mathbf{d}^l, \mathbf{p}^l) = \sum_{\{i,j\} \in \mathcal{A}} [w_1 \cdot \mathbf{1}(|D_i^l - D_j^l| = 1) + w_2 \cdot \mathbf{1}(|D_i^l - D_j^l| > 1)] \qquad (6.5)$$

Here, $\mathbf{1}$ is the indicator function, $\mathcal{A}$ contains sets of neighbours as above, and D_i^l is the

% Missed	Cones	Teddy	Tsukuba	Venus
MAX	71.42253518	71.84294462	34.57740247	62.33148575
MPI_r	19.09491121	22.50036299	6.55560121	5.27275390
MIN	19.06307191	22.48645276	6.52367249	5.20088896
p(RAND beats MPI_r):	0.0392	0.0392	0.4522	0.3302

Table 6.1: Comparison of MPI_r on four stereo matching problems *without* median filtering. Values indicate percent of mislabeled pixels. The last row gives the estimated probability that 100 uniformly randomly acquired points will find a minimum $<$ MPI_r. Maximum absolute error (MPI_r – MIN) was on the 'Teddy' data set at 0.0224, and maximum relative error (MPI_r – MIN)/(MAX – MIN) was 0.0006, on the 'Tsukuba' data set.

% Missed	Cones	Teddy	Tsukuba	Venus
MAX	59.96534228	60.33118962	16.95288271	45.41595279
MPI_r	16.70819967	20.28740048	5.06180450	3.59058305
MIN	16.68799520	20.26502341	5.05496260	3.57062071
p(RAND beats MPI_r):	0.0392	0.0769	0.1480	0.5145

Table 6.2: Comparison of MPI_r on four stereo matching problems *with* median filtering. Values indicate percent of mislabeled pixels. The last row gives the estimated probability that 100 uniformly randomly acquired points will find a minimum $<$ MPI_r. Maximum absolute error (MPI_r – MIN) was on the 'Teddy' data set at 0.0223, and maximum relative error (MPI_r – MIN)/(MAX – MIN) was 0.0006, the 'Tsukuba' data set.

disparity of pixel p_i^l, which is proportional to the inverse of the depth d_i^l; it is common for practitioners to work with disparity values instead of depths, although there is a one-to-one mapping from one to the other using camera geometry. Note that this smoothing function does not depend on the pixel data $\mathbf{p}^l$. The only difference between the two algorithms we examine is that one uses a median filtering technique as a post-processing step, and the other does not. The median filter takes each 3×3 patch in the MAP disparity map, and replaces the center disparity with the median disparity in this patch.

Results Four commonly studied image pairs with corresponding ground-truth data from the Middlebury stereo vision web site [40] were fed into the two algorithms. We performed an exhaustive grid-search using 2500 points to optimize w_1 and w_2 for each of the eight data-algorithm pairs—one exhaustive search takes approximately two hours on a 1.83GHz Intel Core Duo. The performance measure we use is the percentage of pixels in the image whose disparity given by the algorithm differs from the ground truth data by more than 1; disparity is quantized to integer values in $\{0, 1, ..., 255\}$. We report the maximum (worst) and minimum (best) performance values found by this exhaustive search as points of comparison.

Tables 6.1 and 6.2 give the performance values found by exhaustive search and by GP-based global optimization using the MPI_r criterion with $\xi_r = 0.1$ and independent log-

normal priors on length scales. On each data-algorithm pair, the GP-based method was allowed to acquire 100 data points, which takes approximately 3.75 minutes of compute time, again on an Intel Core Duo at 1.83GHz.

We also report an estimate of the probability that the minimum of 100 uniformly randomly sampled points would produce a function value better than that found by MPI_τ. We estimate this by counting the fraction of the 2500 grid points acquired that are below the MPI_τ threshold, and computing the probability that a random sample of 100 points would include at least one of these. In the 'Cones' and 'Teddy' datasets, these probabilities are quite low, indicating that MPI_τ is finding a global minimum that is well-separated from other local minima, and that is unlikely to be attained by chance. On the other hand, on the 'Tsukuba' and 'Venus' datasets, the probabilities are much higher. Inspection of the error surface for these datasets has shown that the 'Cones' and 'Teddy' error surfaces have a pronounced slope around the global minimum, whereas the 'Tsukuba' and 'Venus' error surfaces have a large and very flat area surrounding the global minimum that is full of small, seemingly i.i.d. fluctuations whose values are very near the global minimum. In these last two cases, MPI_τ does not appear to present a significant advantage over random point acquisition.

Despite this, in all of the cases examined, the GP-based method found parameter settings that were worse by no more than 0.3% of the pixels in the image as compared to the exhaustive search. This difference was deemed acceptable given the reduction in total compute time by a factor of 32.

Conclusion These experiments illustrate the use of GP-based optimization in an out-of-the-box fashion on a novel problem. The specific technique used—MPI_τ with $\xi_\tau = 0.1$ and independent log-normal priors on length scales—was suggested by the extensive experimental analysis from Chapter 5. The results obtained are important to practitioners researching stereo matching algorithms, because they demonstrate a mechanism for investigating the tuning of stereo matching parameters that takes far less computation time than other naïve approaches. Our approach also makes more rigorous parameter optimization possible: Since so few function evaluations are needed, it may be useful and feasible in the future to optimize the error of algorithms on several sets of data at once, resulting in parameter settings that are more generally applicable. Such a study, which would have taken days to run previously, could be completed in hours using our approach.

Chapter 7

Conclusion

This thesis has sought to make GP-based methods a practical choice for the task of global function optimization. To this end, we have made several advances to the GP-based approach, made an extensive empirical evaluation, and presented two successful real-world applications. We now summarize these advances and present avenues of further research in the area.

7.1 Contributions

We developed and used a new methodology for evaluating global optimization techniques based on generating many test functions and evaluating performance on each. Having this method of evaluation enabled us to study the effect of adapting kernel parameters and changing the exploration parameter ξ_r. Our empirical evaluation shows that in most cases no tuning of ξ_r is required at all.

The parameter ξ_r controls the exploration level of the new acquisition criteria that we developed in Chapter 3. These acquisition criteria are invariant to vertical shifting and scaling of the objective function, so our results hold not only for all of our test functions, but for all vertically shifted and scaled versions of the test functions. This reduces the need to tune ξ_r to different objective functions. Our empirical evaluation showed that in the cases we examined, little no tuning of ξ_r is required, particularly for MEI_r.

We demonstrated in Chapter 5 that using a MAP objective can be used to reliably learn the kernel parameters, eliminating the need for pre-acquisition of the function and again reducing the need for user input. In the process of developing the new EEC prior, we gave a novel polynomial time algorithm in Chapter 2 for computing the expected Euler characteristic of axis-scaled isotropic Gaussian processes over closed intervals of $\mathbb{R}^d$.

Finally, We illustrated the use of gradient information with GP optimization, showing in particular that MEI_τ can perform very well when given this information. In fact our results show that MEI_τ with gradient information can perform much better than the BFGS algorithm with random restarts when given the same number of function evaluations.

7.2 Future Work

Recently, many advances have been made by the machine learning community that improve Gaussian process-based methods, particularly in improving computational efficiency and developing more flexible models. Fortunately, most of these methods are "plug-and-play" with the approaches described in this work since the acquisition criteria can be used with any underlying GP model. We present two methods for improving computational efficiency that could be integrated into the optimization methods we have described.

We also foresee a wealth of interesting research arising from applying GP-based optimization to different problems in different application areas. There are many problems in computing science and in other disciplines that are solved by optimizing a "surrogate" function that presents a more tractable objective than the original problem; we present machine translation as one example.

7.2.1 Reducing Computational Requirements

Perhaps the biggest obstacle to more widespread use of GP-based optimization is the computational expense involved in computing the next acquisition location. In GP based optimization to date, only naïve methods that take time cubic in the number of data points have been used for computing posterior distributions. This restricts the applicability of GP based methods to problems where the cost of function evaluation makes that degree of overhead acceptable, i.e. problems where function evaluation is very expensive and/or the number of data points considered is small. However, several approaches have been developed that approximate the posterior distribution using less computation.

Subset of Data Points Approximation The simplest approximation one can make when there are too many data points to deal with is to simply ignore some of the data. A small subset of m data points can be retained, and computation times are reduced accordingly since only m kernel functions are used to construct the posterior.

The performance of this technique depends on the complexity of the posterior in terms

of how many points are needed to approximate it well. Also, the selection of *which* points to use is important. Lawrence et al. [26] suggest greedily choosing the m points where posterior variance is highest, which is equivalent to adding points that give the greatest reduction in the entropy of the posterior.

Projected Process Approximation The Projected Process Approximation [36] is a mechanism for approximating the posterior process with $m < n$ kernel functions but still using all n observed data points. Suppose $\mathbf{x}_m$ contains the m points where we will center the kernels used for the approximation. Then

$$\begin{aligned}
\mu(F_z|F_{\mathbf{x}} = \mathbf{f}) &\approx \mu_0(z) \\
&\quad + k(z, \mathbf{x}_m)(k(\mathbf{x}_m, \mathbf{x})\, k(\mathbf{x}, \mathbf{x}_m) + \sigma_n^2 k(\mathbf{x}_m, \mathbf{x}_m))^{-1} (\mathbf{f} - \mu_0(\mathbf{x})) \\
\sigma^2(F_z|F_{\mathbf{x}} = \mathbf{f}) &\approx k(z, z) - k(z, \mathbf{x}_m) k(\mathbf{x}_m, \mathbf{x}_m)^{-1} k(\mathbf{x}_m, z) \\
&\quad + \sigma_n^2 k(\mathbf{x}_m, \mathbf{x})(\sigma_n^2 k(\mathbf{x}_m, \mathbf{x}_m) + k(\mathbf{x}_m, \mathbf{x})\, k(\mathbf{x}, \mathbf{x}_m))^{-1} k(\mathbf{x}_m, z)
\end{aligned}$$

Note that the posterior mean now involves only m kernel evaluations between the query point z and the subset of data $\mathbf{x}_m$, and the matrices that need to be inverted are of size $m \times m$. Compute time for inference is $\mathcal{O}(m^2 n)$ initial setup, $\mathcal{O}(m)$ time for a posterior mean and $\mathcal{O}(m^2)$ time to compute a posterior variance. Like the Subset of Data Points technique, m kernel functions are used to model the posterior, but the approximation is fit to *all* the data we have instead of just the points in $F_{\mathbf{x}_m}$. As before, however, the choice of which points to include can make a significant difference in the quality of the approximation. A greedy criterion for including points again based on the posterior variance is given by Csató and Opper [10].

Each of these approximation methods can be quite simply "dropped in" to the GP-based optimization framework described in this thesis; however the effect on performance of making this type of approximation still needs to be assessed.

7.2.2 Application to Translation Directed Word Alignment

The field of statistical machine translation has been dominated by the approach of first solving the *word alignment* problem [8]. Given a sentence and its translation in a different language, a word alignment is a set of word pairs where the first word comes from original sentence and second word from the translation. In a "good" word alignment, these pairs (or "links") will exist if and only if the words are (possibly partial) translations of one

another. In a statistical machine translation system, word aligner is typically trained on parallel text, and is then used with what is known as a "decoder" to construct translations of new sentences. To date, however, word aligners have not been trained using translation quality as the objective function, because of the complex interactions between the aligner and the decoder, and because evaluating modifications to the word aligner is very time consuming. They are instead trained on a simpler objective based on what humans consider to be "good" alignments.

It is possible in principle to optimize the various parameters that direct the actions of a word aligner using translation quality as our objective. The word aligner is controlled by about 10 real input parameters, and can automatically provide us with a measure of translation quality. Since this problem has very expensive function evaluations and since the mapping is not well understood, we anticipate GP-based optimization to be a good candidate for finding good solutions and providing insight into the objective function in terms of how different parameters affect performance.

7.3 Summary

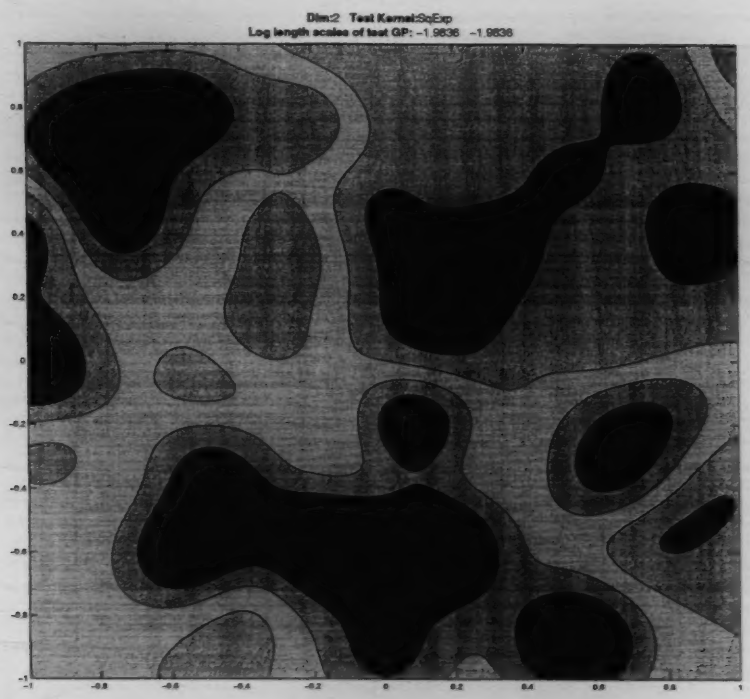
In this thesis we have described, tested, and improved upon Gaussian process-based global optimization. Our work improves not only the performance of the technique, but the breadth of problems to which it is applicable. We have shown two applications where the technique has proven useful, and we expect that further usefulness will be revealed in the future.

Appendix A

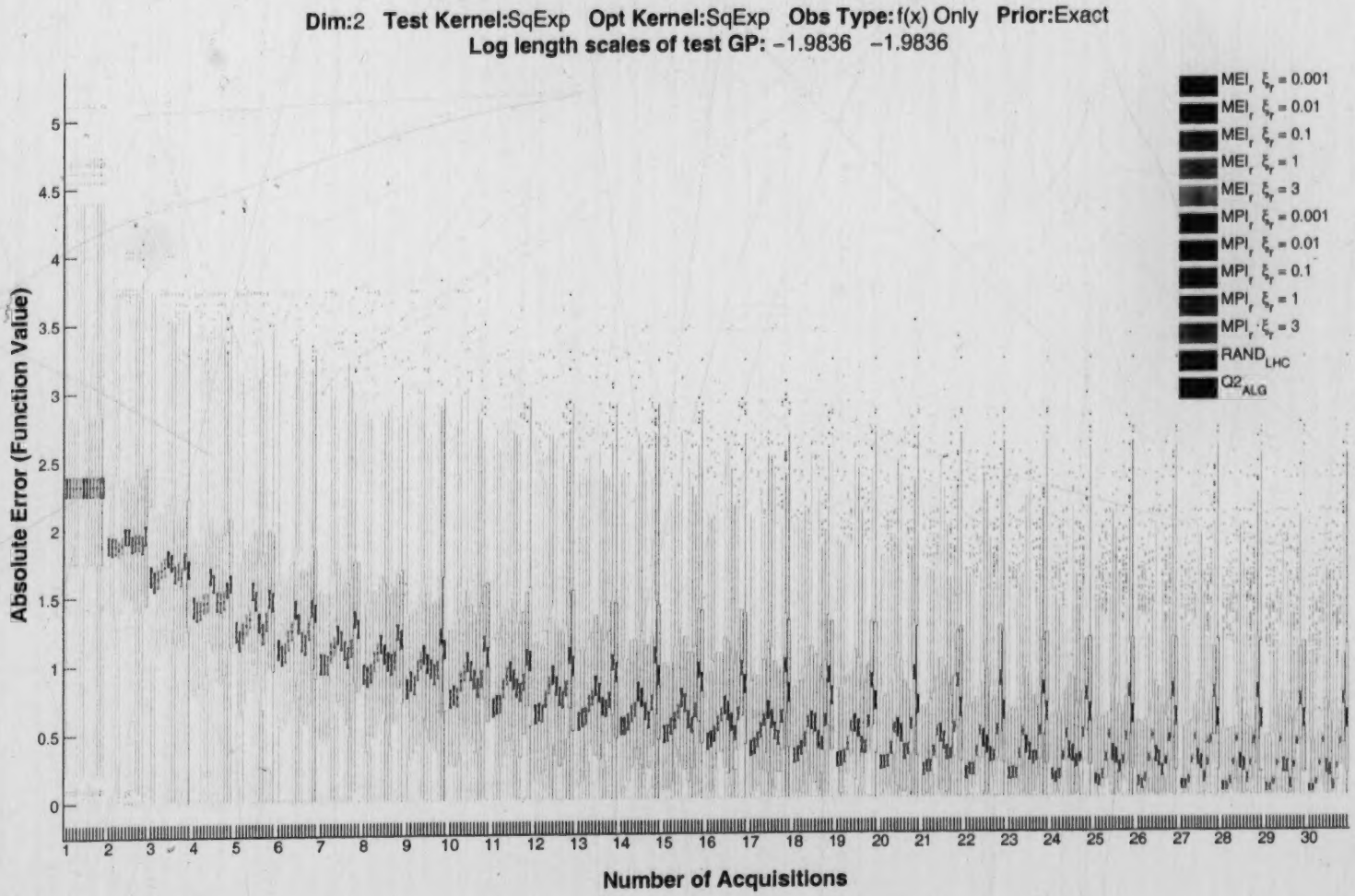
Complete Experimental Data

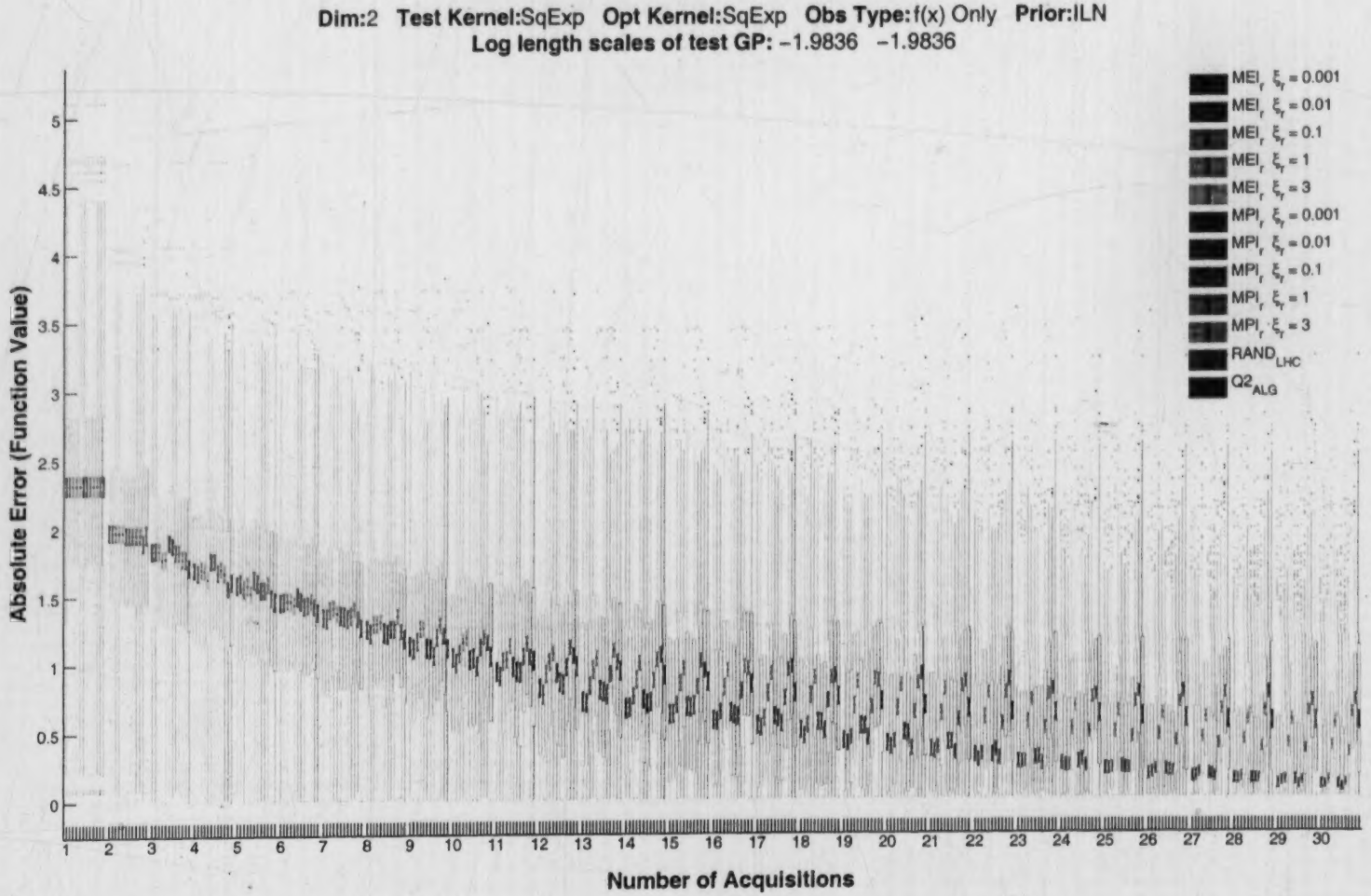
A.1 Results using 2D squared exponential test kernel, equal length scales

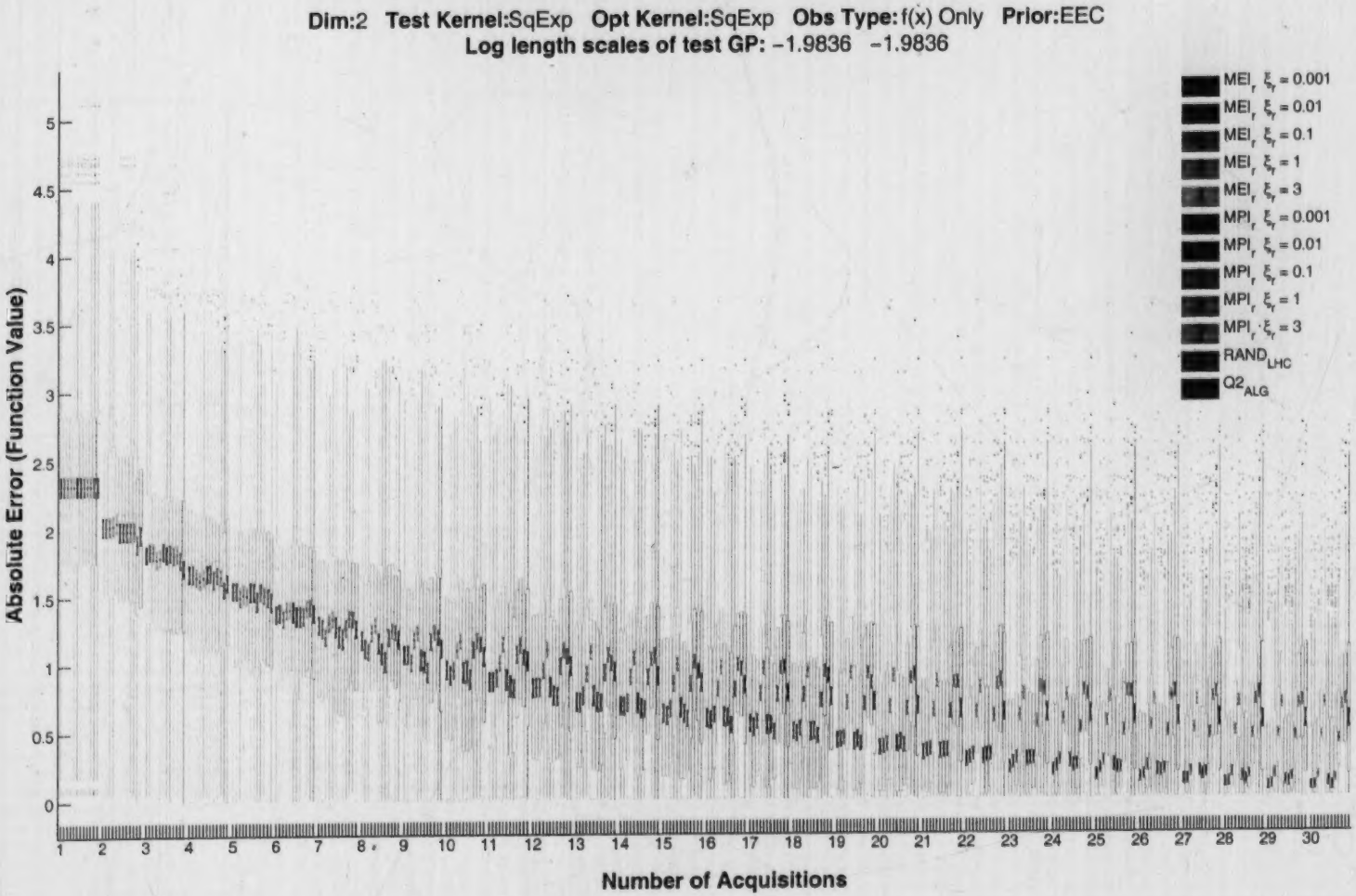
An example posterior mean:

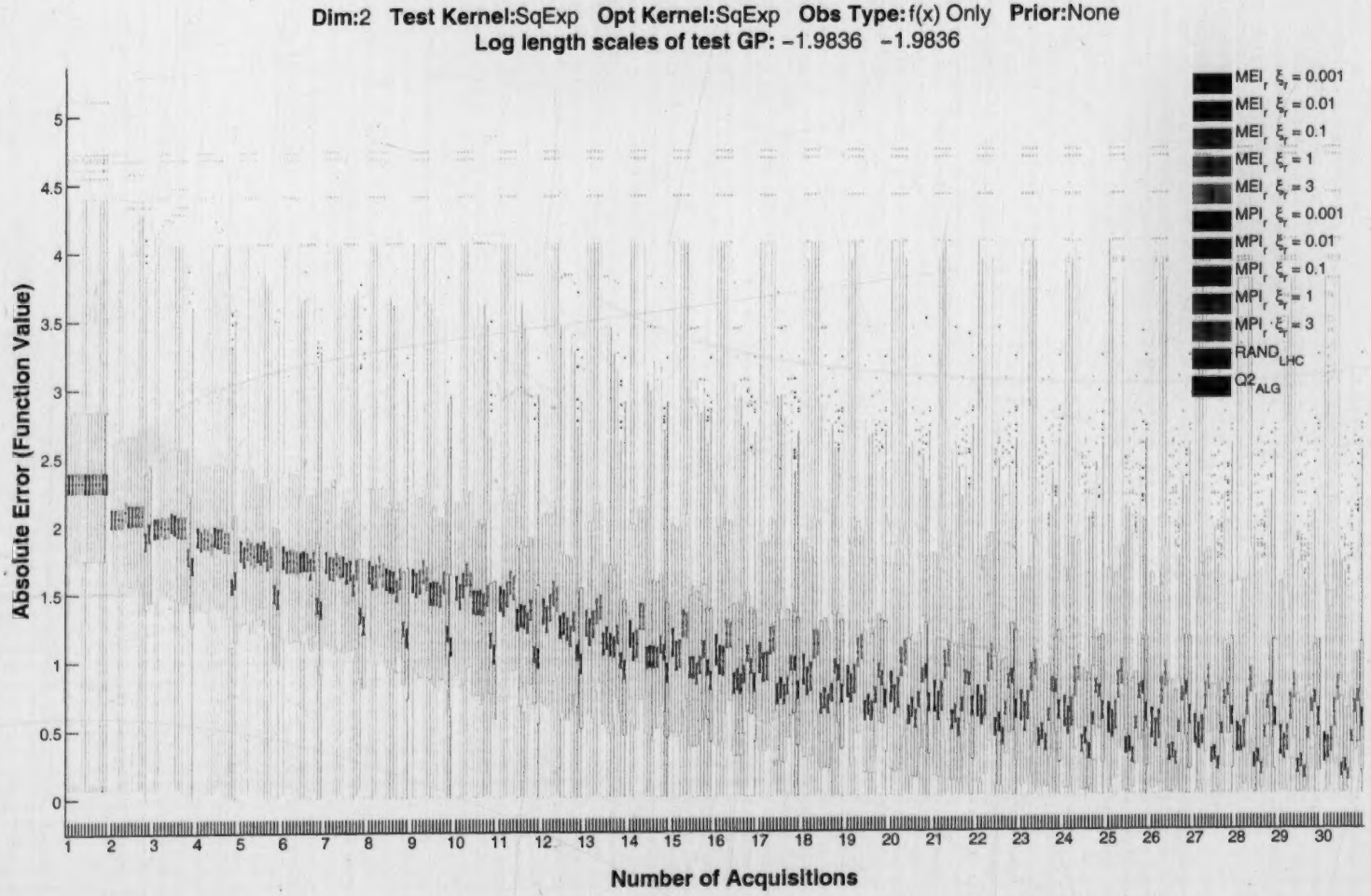


2D squared exponential test kernel with equal length scales. The optimization model also uses a squared exponential kernel. Only function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: −1.9836 −1.9836
MEI_r ξ_Y = 0.001
MEI_r ξ_Y = 0.01
MEI_r ξ_Y = 0.1
MEI_r ξ_Y = 1
MEI_r ξ_Y = 3
MPI_r ξ_Y = 0.001
MPI_r ξ_Y = 0.01
MPI_r ξ_Y = 0.1
MPI_r ξ_Y = 1
MPI_r ξ_Y = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

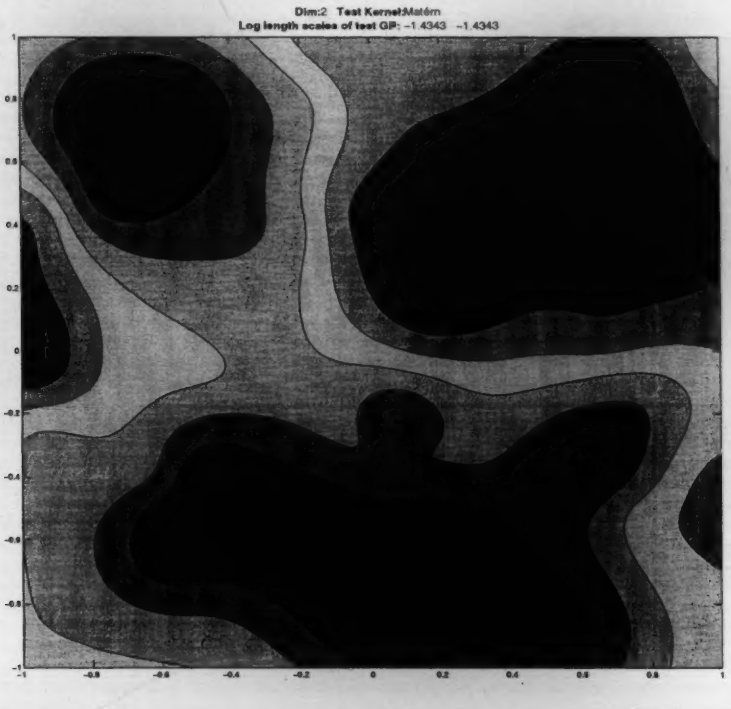

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) Only Prior:ILN
Log length scales of test GP: -1.9836 -1.9836
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
Number of Acquisitions


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:EEC
Log length scales of test GP: -1.9836 -1.9836
MEI, ξ_r = 0.001
MEI, ξ_r = 0.01
MEI, ξ_r = 0.1
MEI, ξ_r = 1
MEI, ξ_r = 3
MPI, ξ_r = 0.001
MPI, ξ_r = 0.01
MPI, ξ_r = 0.1
MPI, ξ_r = 1
MPI, ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

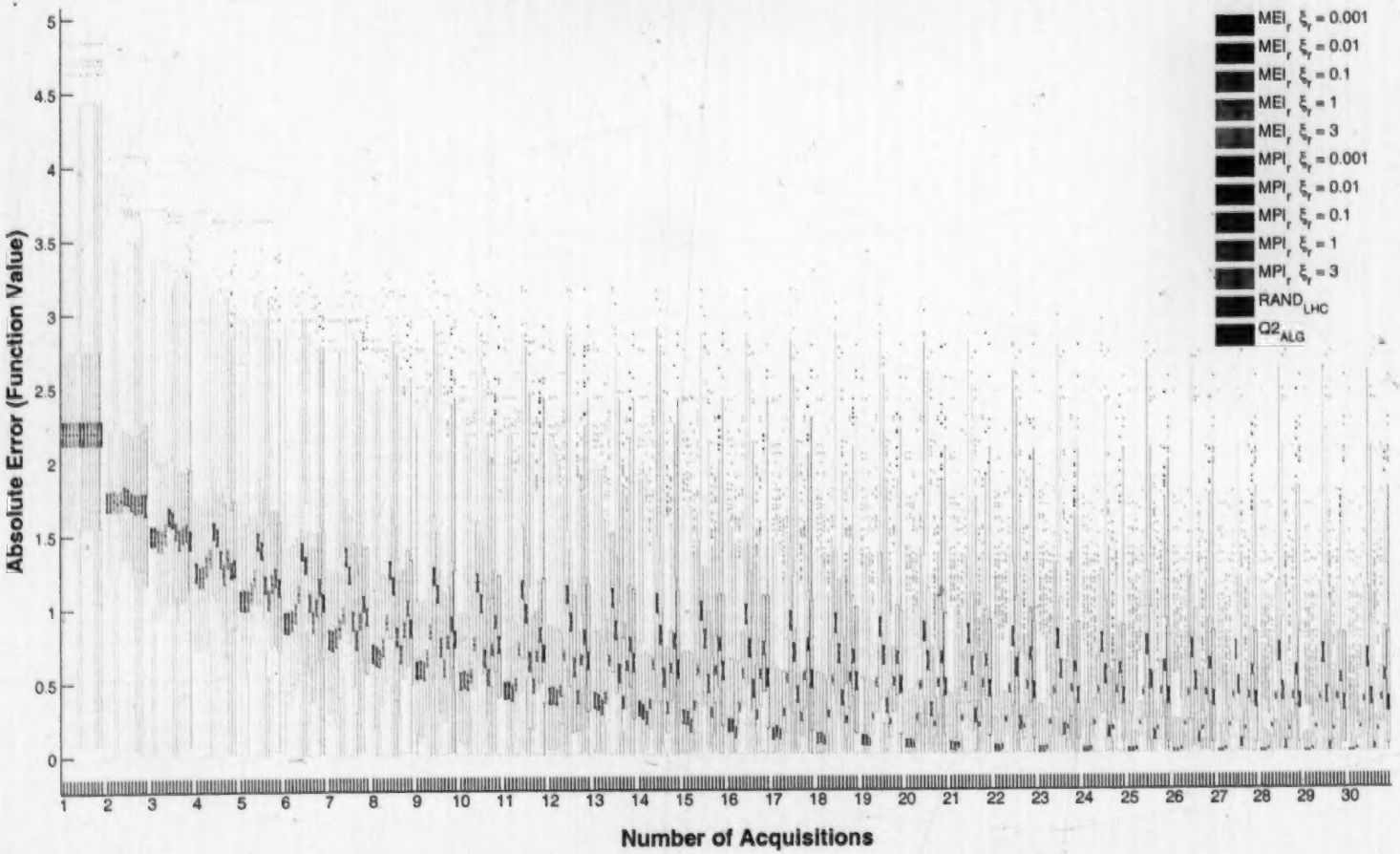

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) Only Prior:None
Log length scales of test GP: -1.9836 -1.9836
Absolute Error (Function Value)
Number of Acquisitions
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Q2 ALG

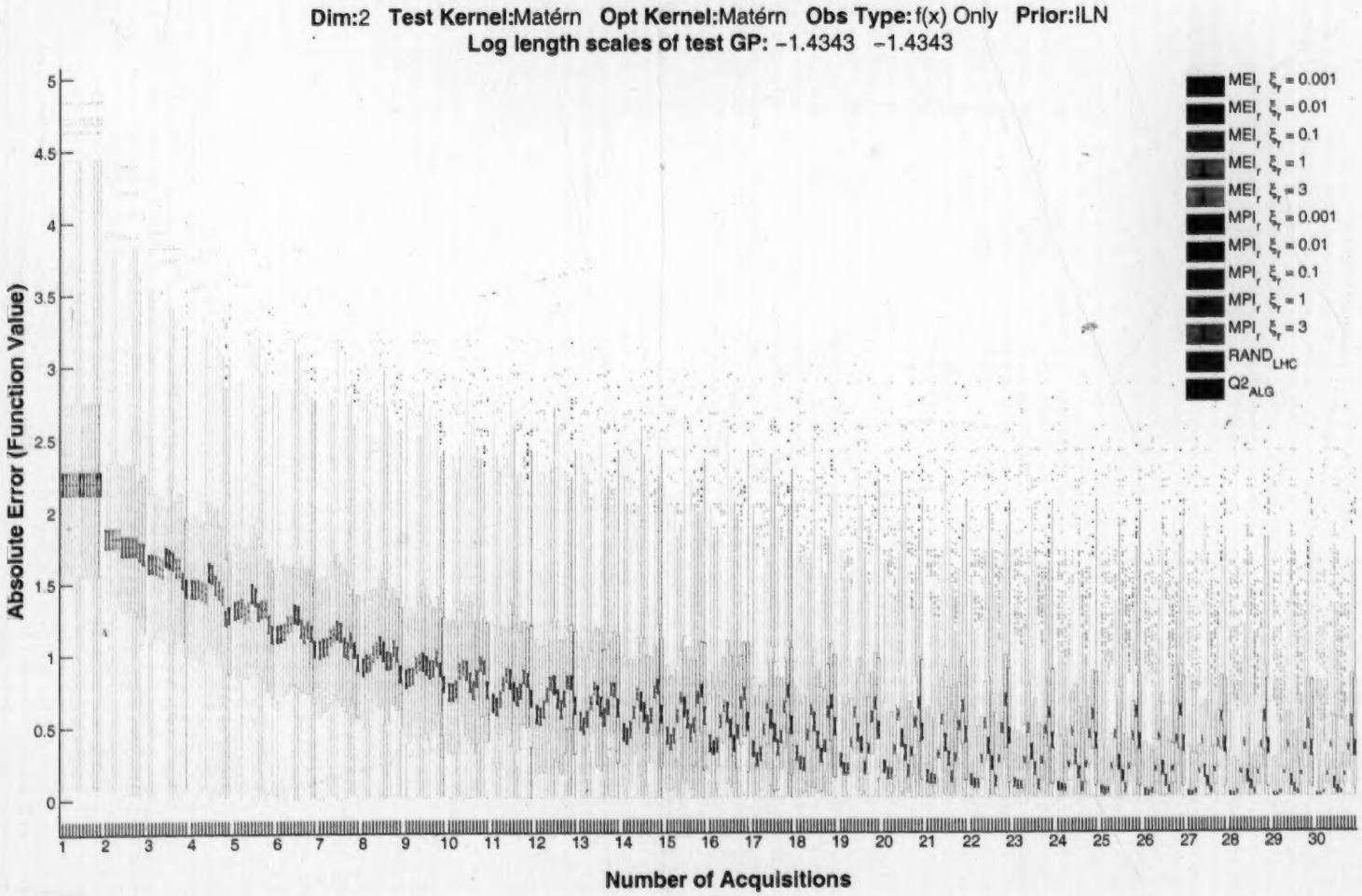
A.2 Results using 2D Matérn test kernel, equal length scales

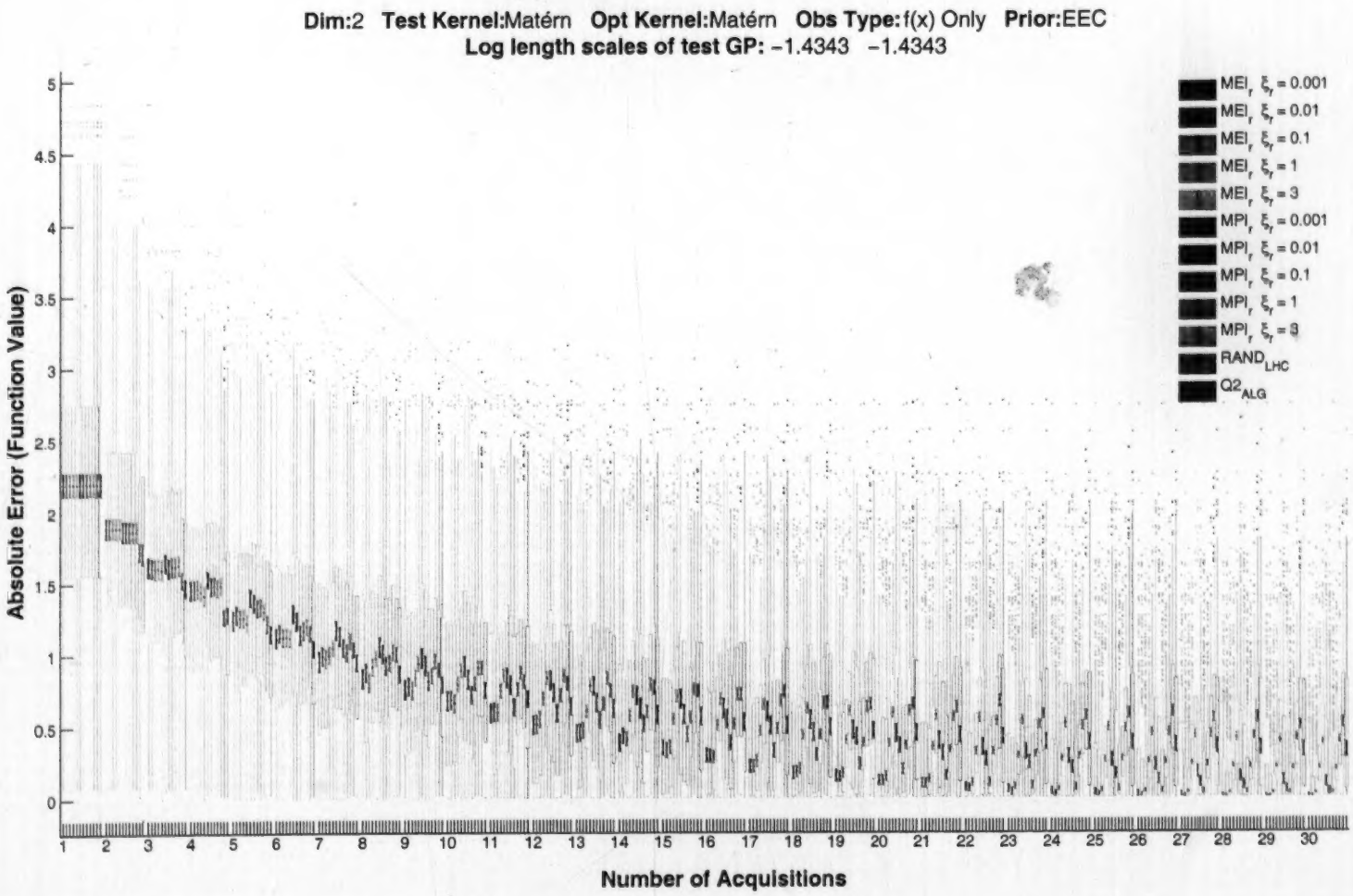
An example posterior mean:

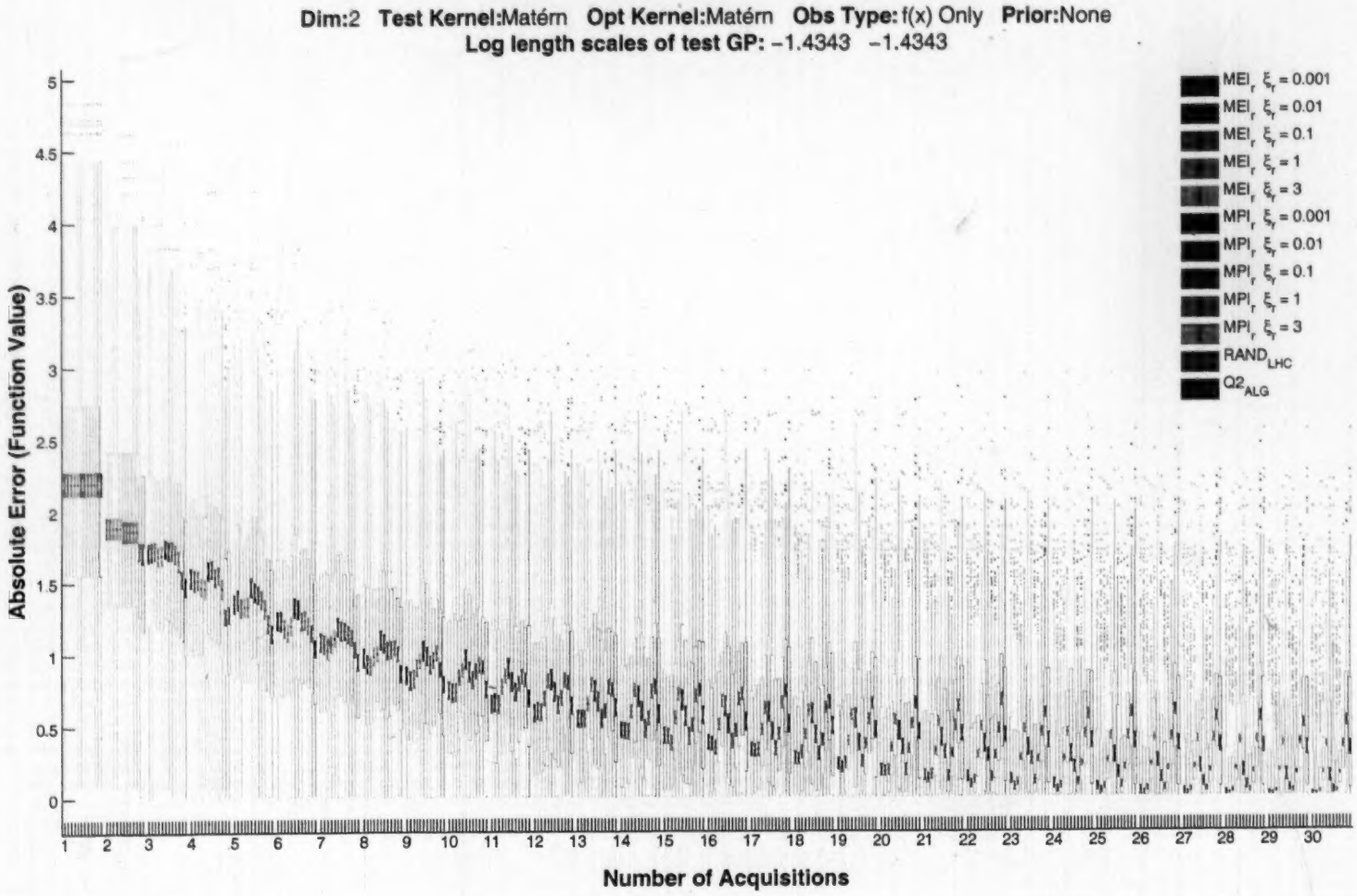


2D Matérn kernel with equal length scales. The optimization model also uses a Matérn kernel. Only observed function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: −1.4343 −1.4343
Absolute Error (Function Value)
Number of Acquisitions
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG

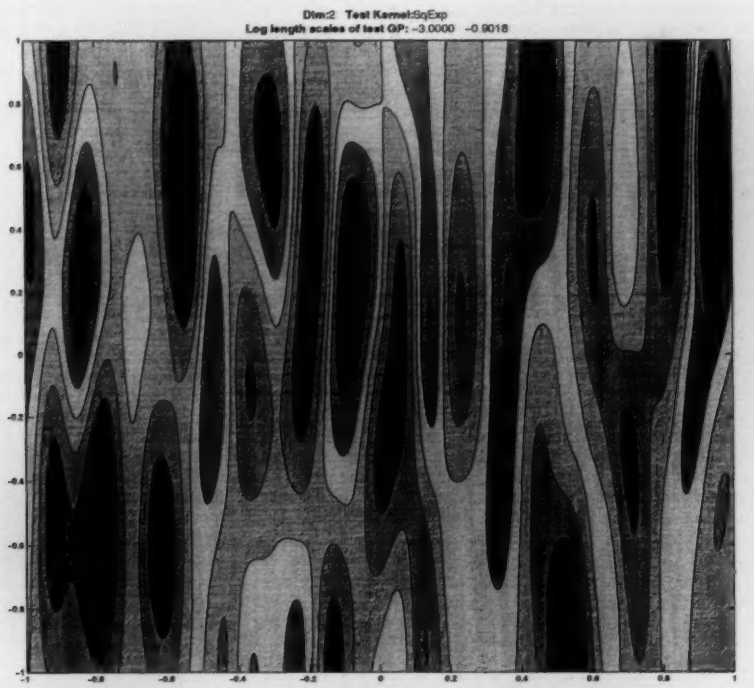

Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) Only Prior:ILN
Log length scales of test GP: -1.4343 -1.4343
Absolute Error (Function Value)
Number of Acquisitions
MEI, ξr = 0.001
MEI, ξr = 0.01
MEI, ξr = 0.1
MEI, ξr = 1
MEI, ξr = 3
MPI, ξr = 0.001
MPI, ξr = 0.01
MPI, ξr = 0.1
MPI, ξr = 1
MPI, ξr = 3
RAND LHC
Q2 ALG


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type:f(x) Only Prior:EEC
Log length scales of test GP: -1.4343 -1.4343
Absolute Error (Function Value)
Number of Acquisitions
MEI, ξ_γ = 0.001
MEI, ξ_γ = 0.01
MEI, ξ_γ = 0.1
MEI, ξ_γ = 1
MEI, ξ_γ = 3
MPI, ξ_γ = 0.001
MPI, ξ_γ = 0.01
MPI, ξ_γ = 0.1
MPI, ξ_γ = 1
MPI, ξ_γ = 3
RAND_LHC
Q2_ALG

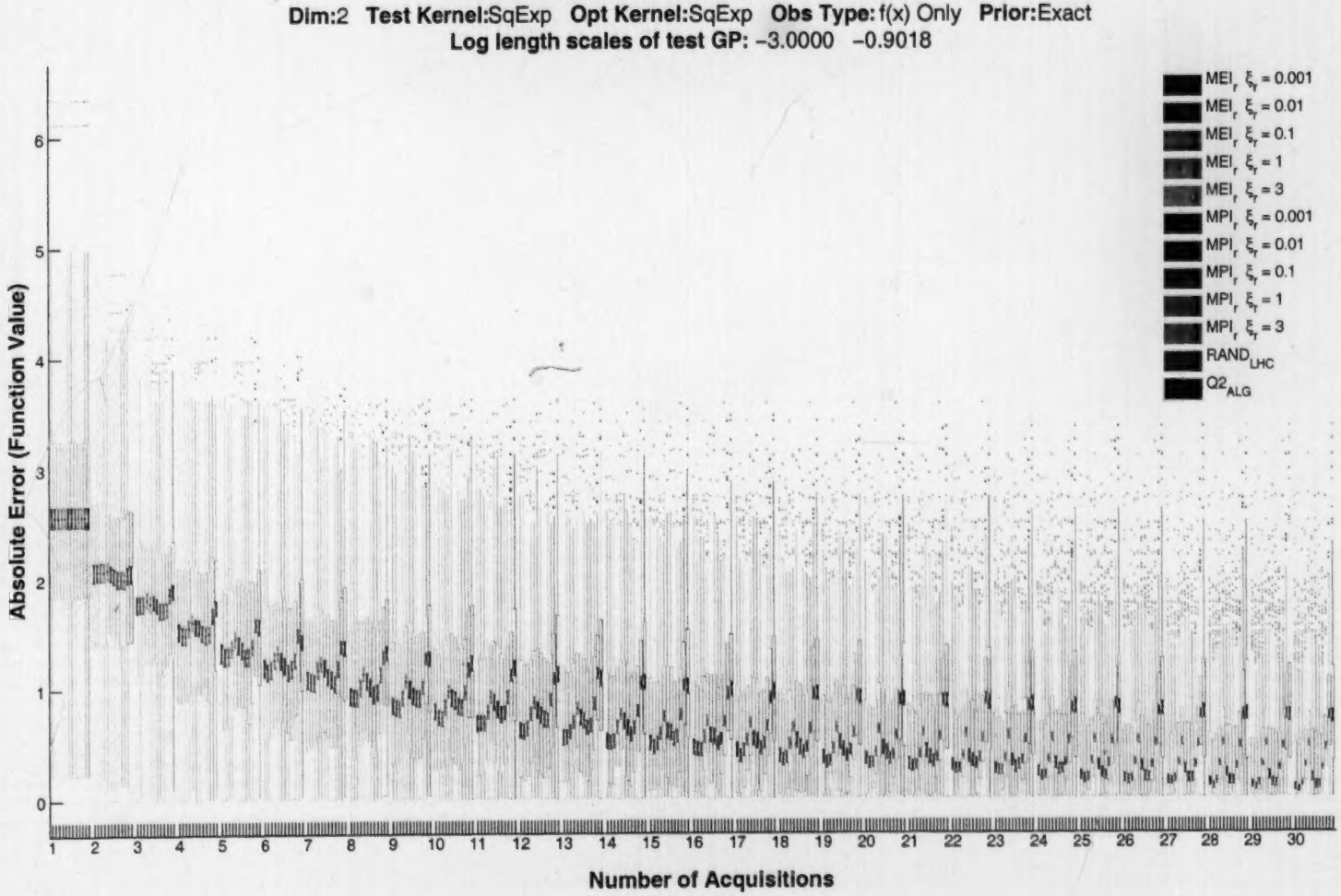

Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) Only Prior:None
Log length scales of test GP: −1.4343 −1.4343
MEI, ξT = 0.001
MEI, ξT = 0.01
MEI, ξT = 0.1
MEI, ξT = 1
MEI, ξT = 3
MPI, ξT = 0.001
MPI, ξT = 0.01
MPI, ξT = 0.1
MPI, ξT = 1
MPI, ξT = 3
RAND LHC
Q2 ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

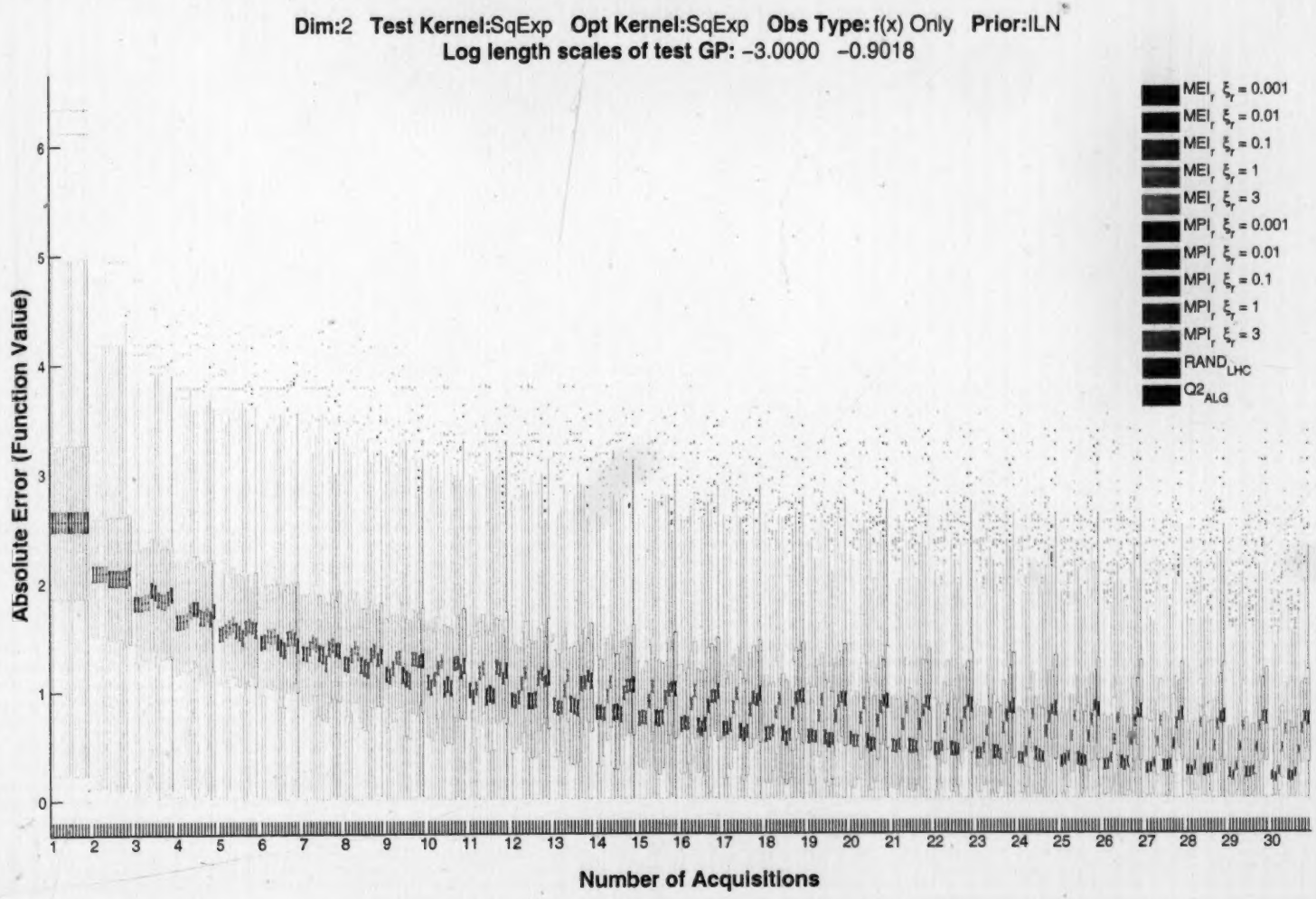
A.3 Results using 2D squared exponential test kernel, unequal length scales

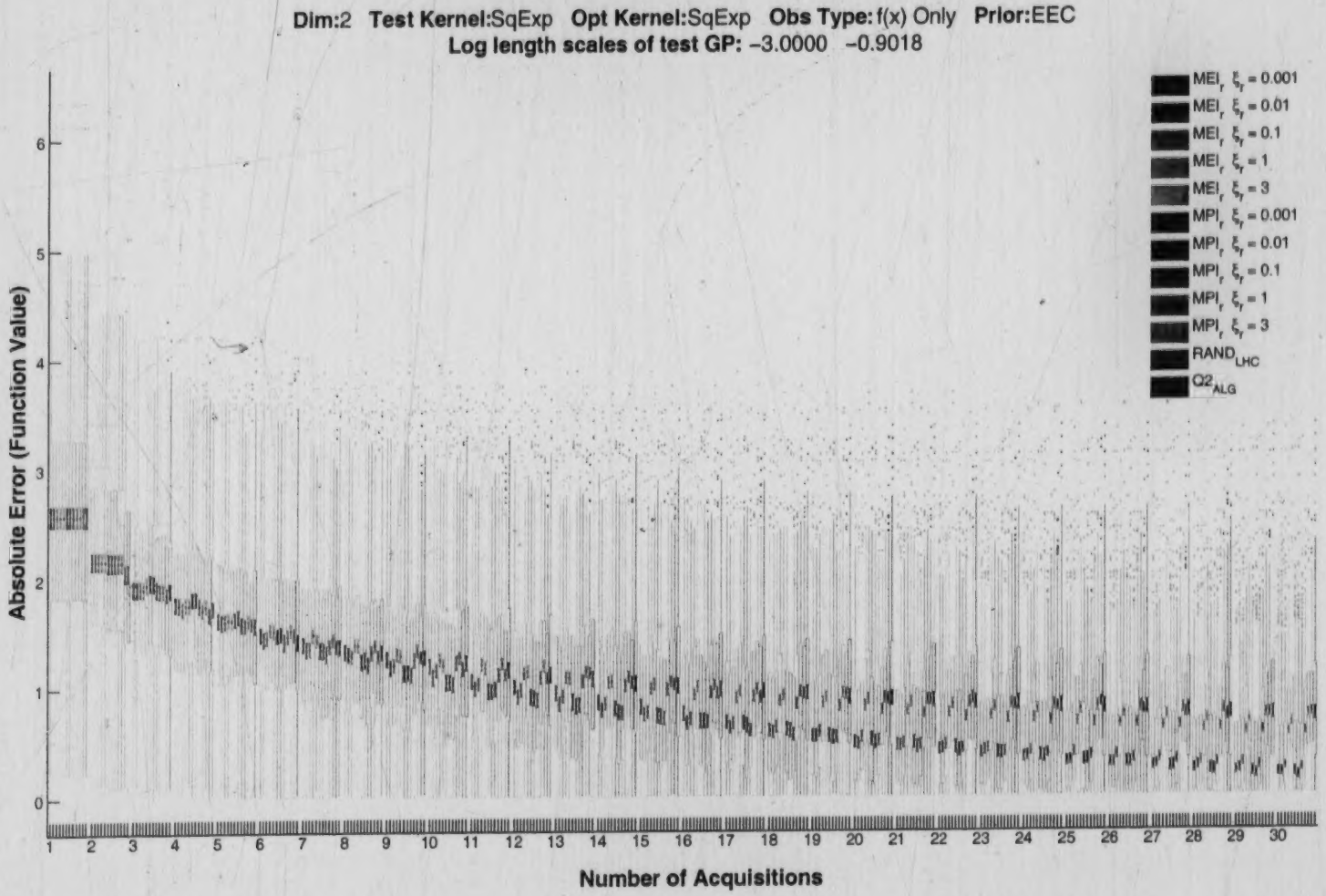
An example posterior mean:

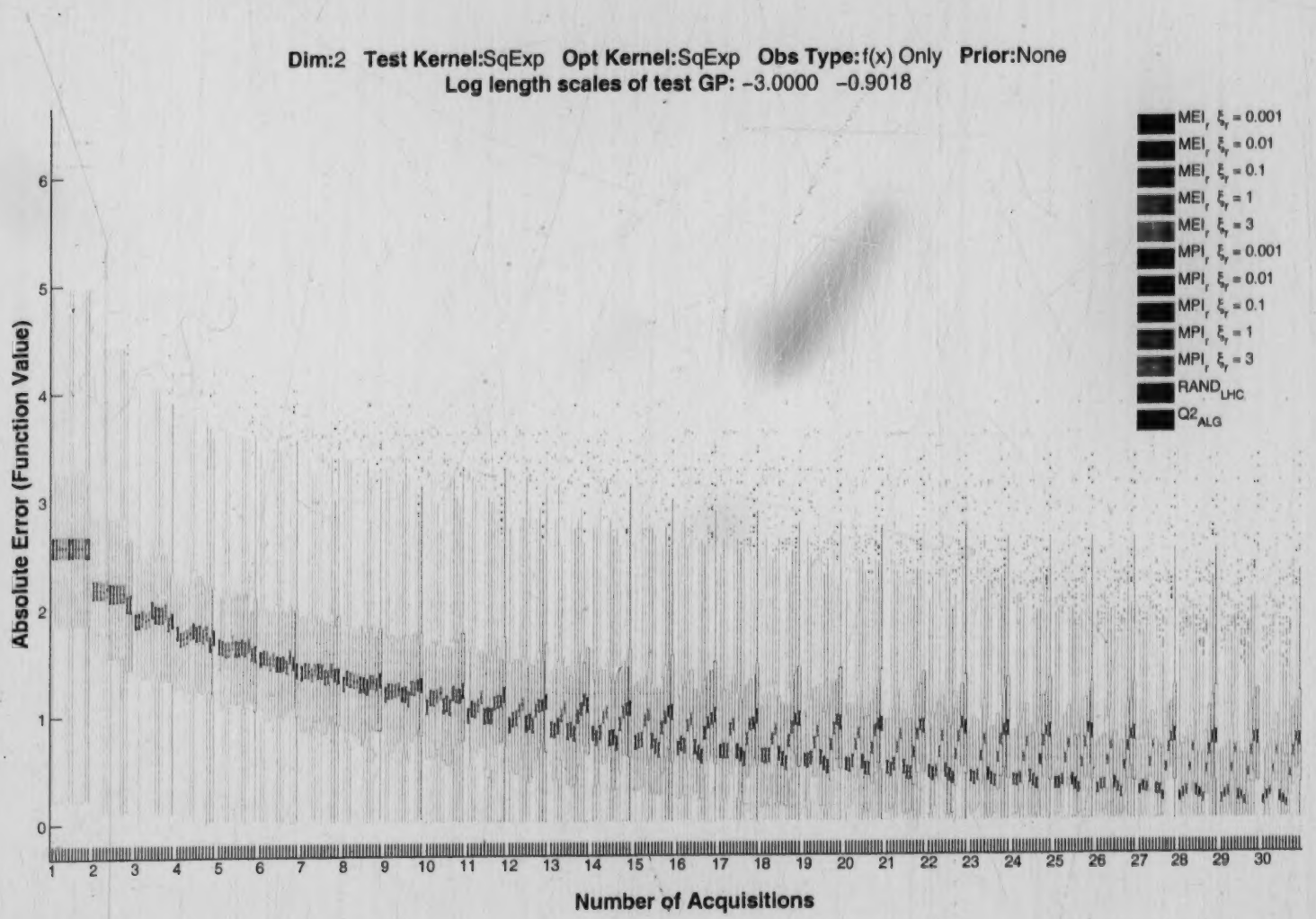


2D squared exponential test kernel with unequal length scales. The optimization model also uses a squared exponential kernel. Only observed function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: -3.0000 -0.9018
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

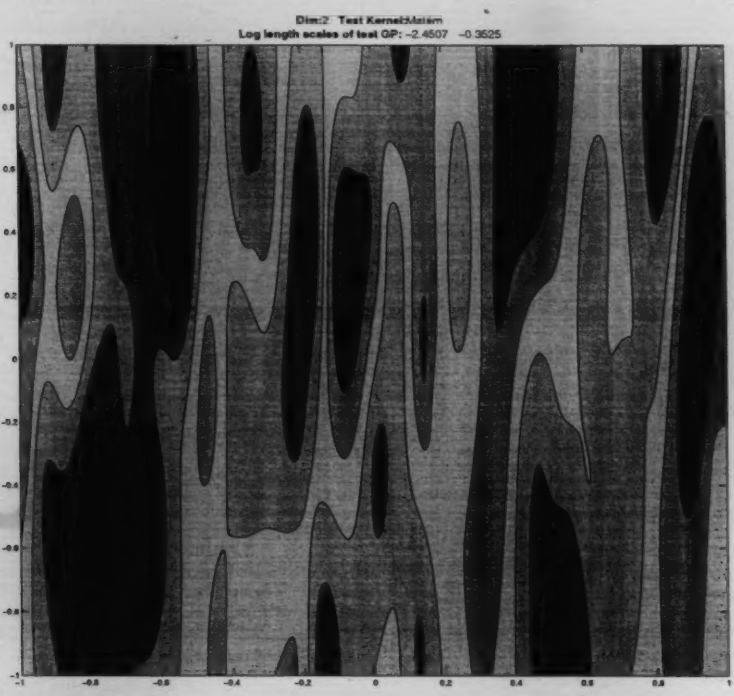

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:ILN
Log length scales of test GP: -3.0000 -0.9018
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
Number of Acquisitions


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:EEC
Log length scales of test GP: -3.0000 -0.9018
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

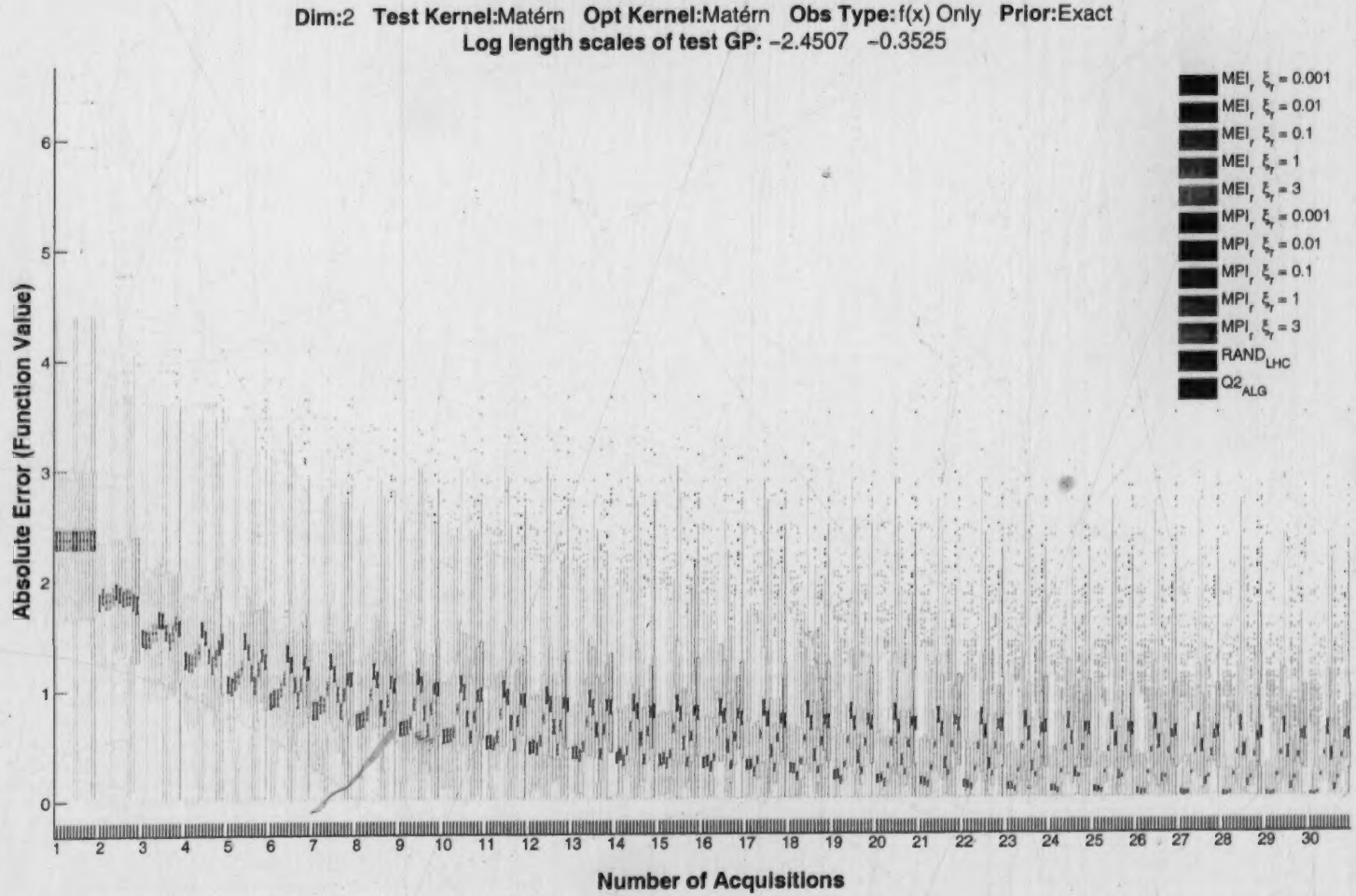

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) Only Prior:None
Log length scales of test GP: -3.0000 -0.9018
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

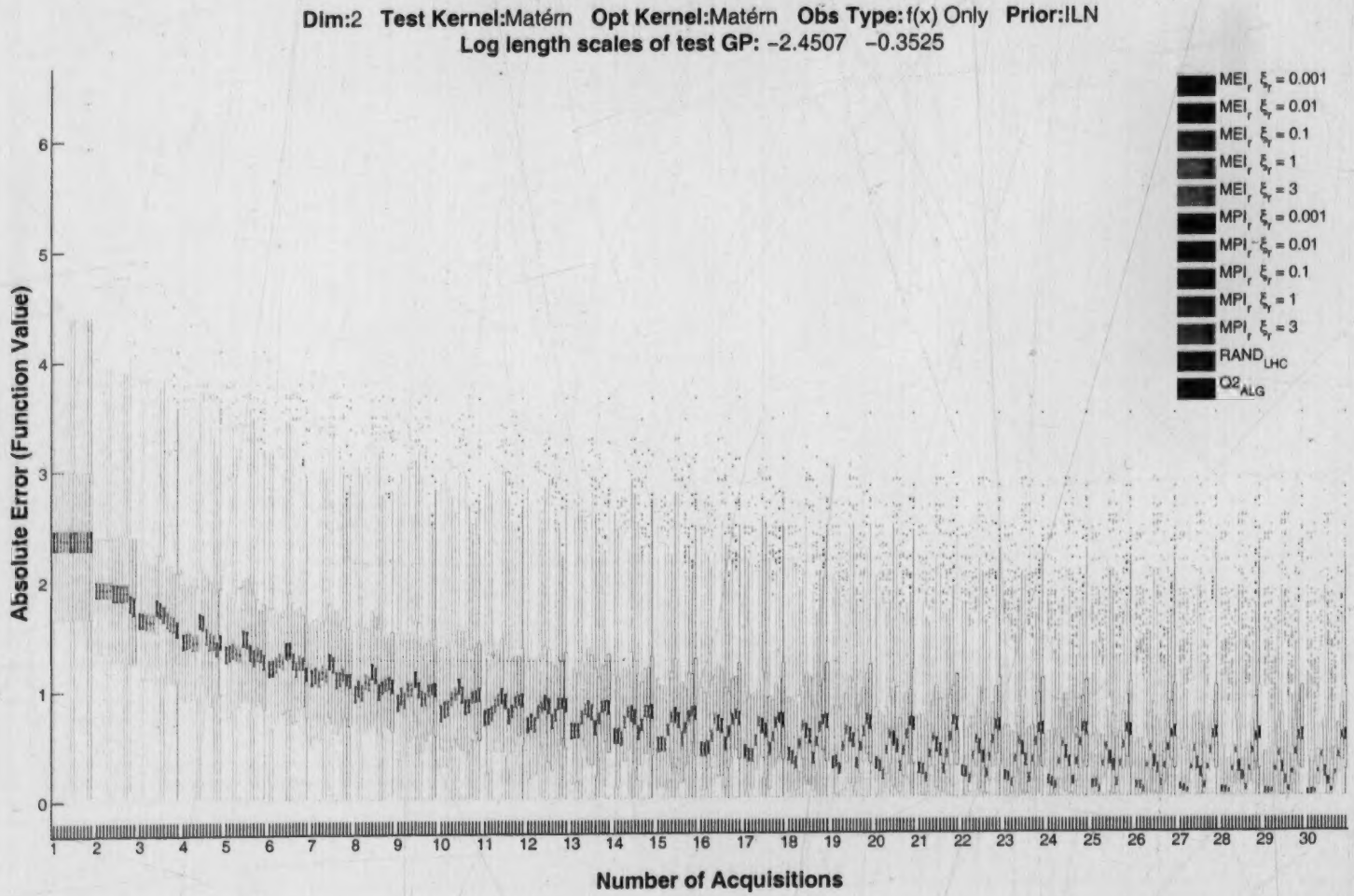
A.4 Results using 2D Matérn test kernel, unequal length scales

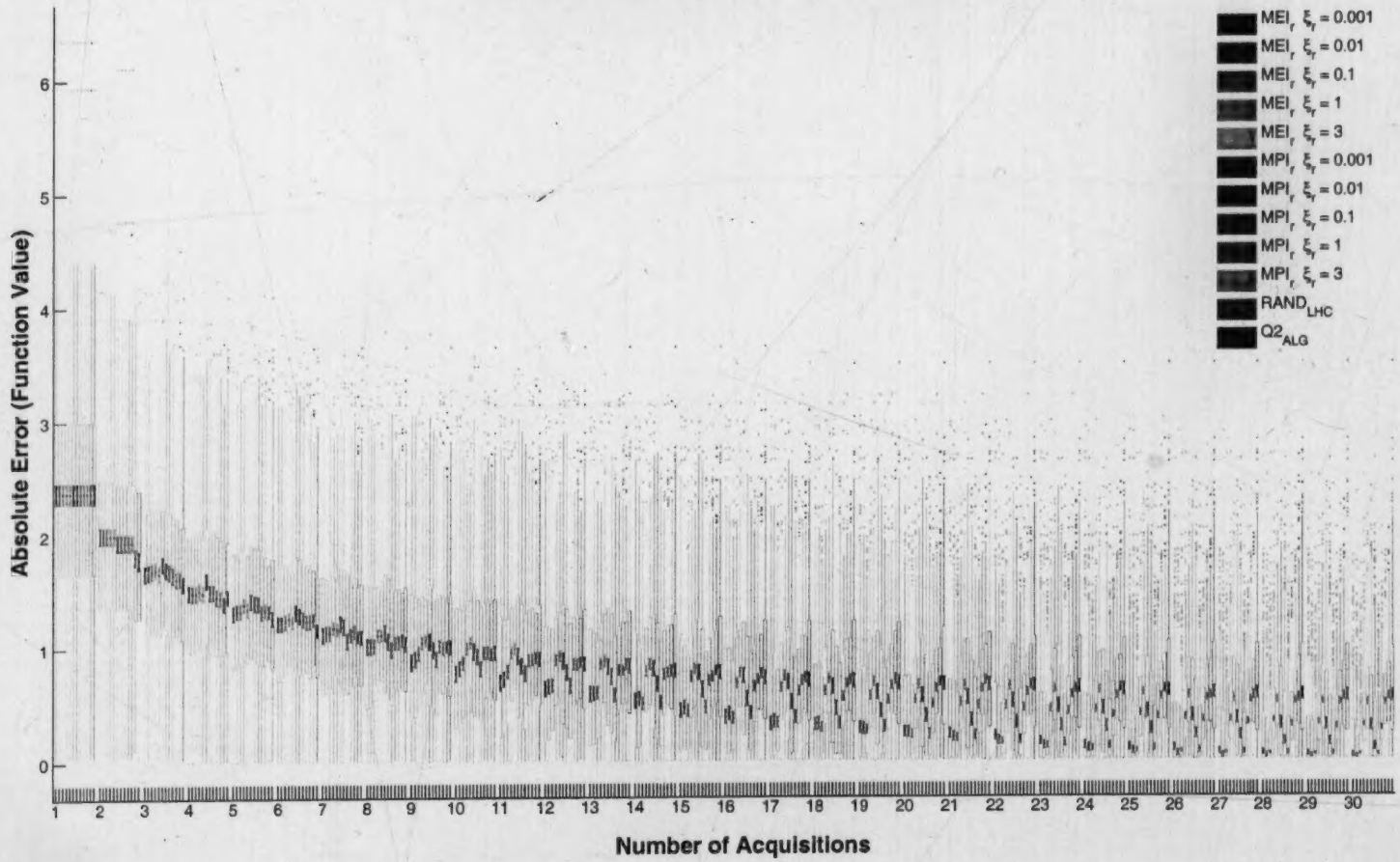
An example posterior mean:

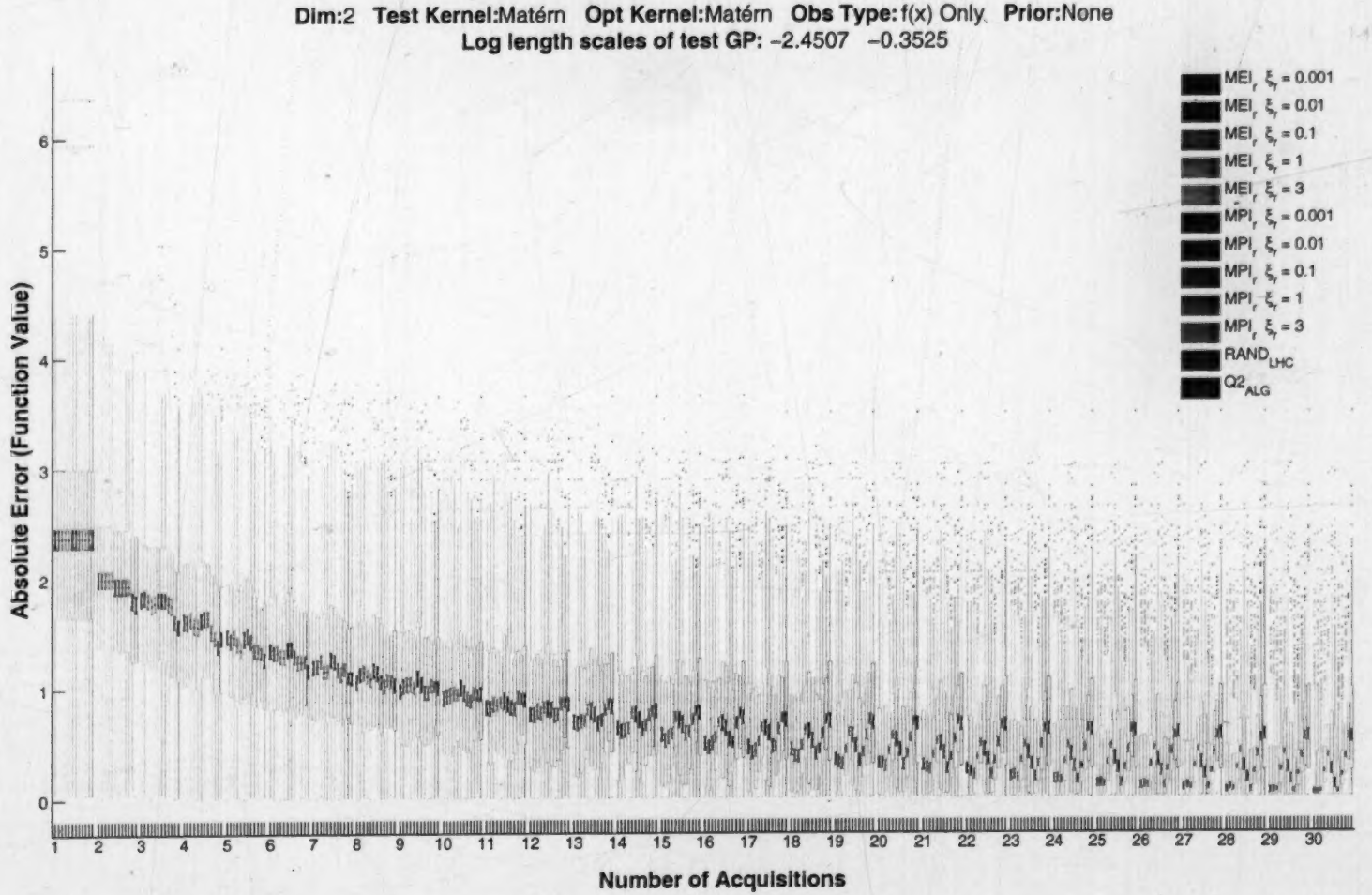


2D Matérn test kernel with unequal length scales. The optimization model also uses a Matérn kernel. Only observed function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type:f(x) Only Prior:Exact
Log length scales of test GP: -2.4507 -0.3525
MEI, ξ_T = 0.001
MEI, ξ_T = 0.01
MEI, ξ_T = 0.1
MEI, ξ_T = 1
MEI, ξ_T = 3
MPI, ξ_T = 0.001
MPI, ξ_T = 0.01
MPI, ξ_T = 0.1
MPI, ξ_T = 1
MPI, ξ_T = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

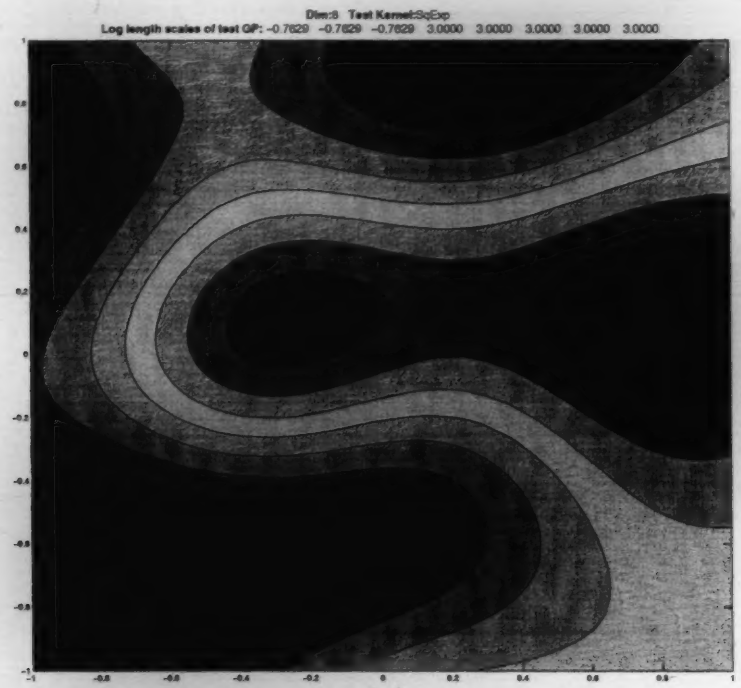

Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type:f(x) Only Prior:ILN
Log length scales of test GP: -2.4507 -0.3525
MEI, ξT = 0.001
MEI, ξT = 0.01
MEI, ξT = 0.1
MEI, ξT = 1
MEI, ξT = 3
MPI, ξT = 0.001
MPI, ξT = 0.01
MPI, ξT = 0.1
MPI, ξT = 1
MPI, ξT = 3
RANDLHC
Q2ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type:f(x) Only Prior:EEC
Log length scales of test GP: −2.4507 −0.3525
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

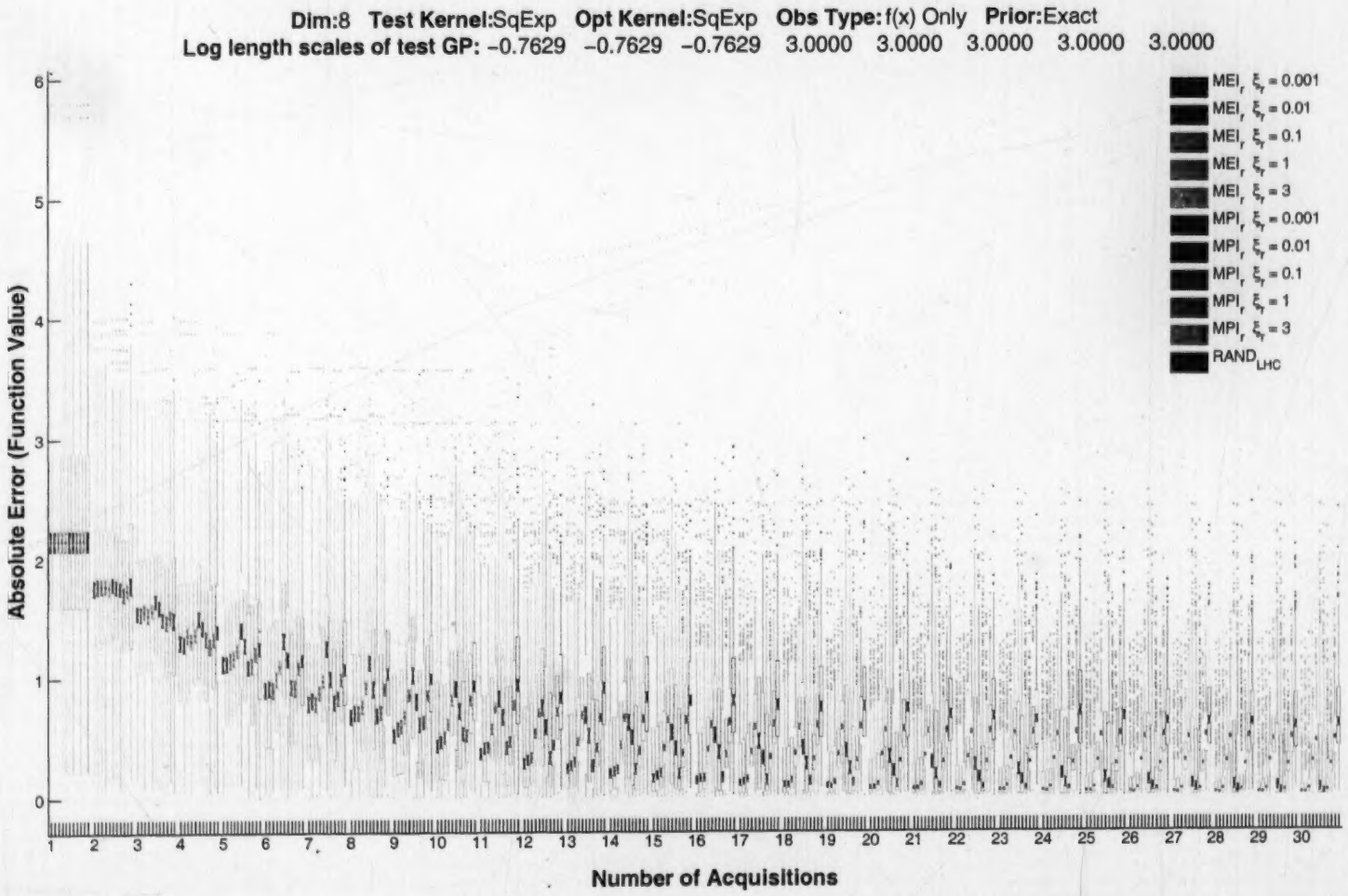

Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) Only Prior:None
Log length scales of test GP: -2.4507 -0.3525
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

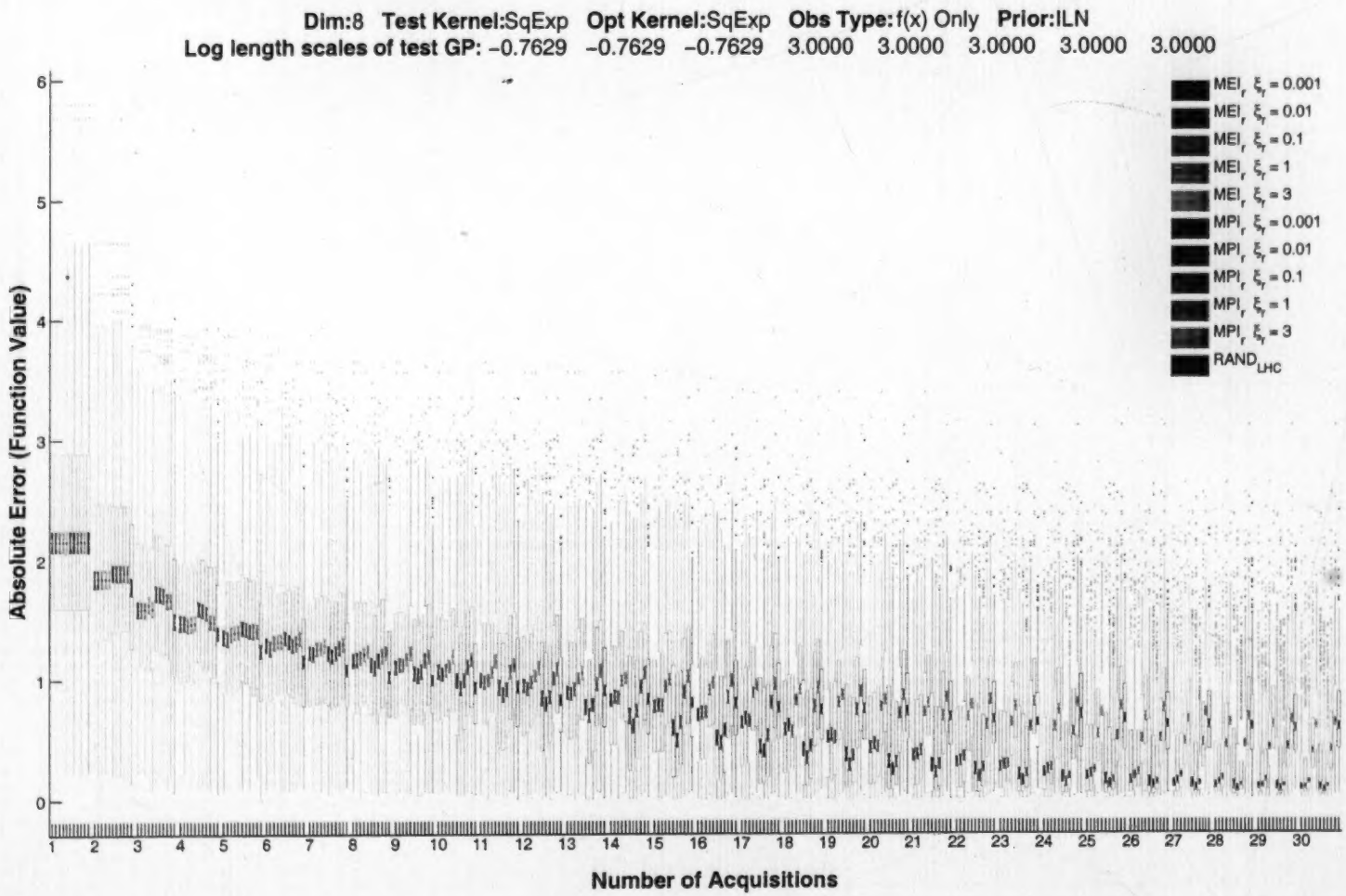
A.5 Results using 8D squared exponential test kernel, unequal length scales

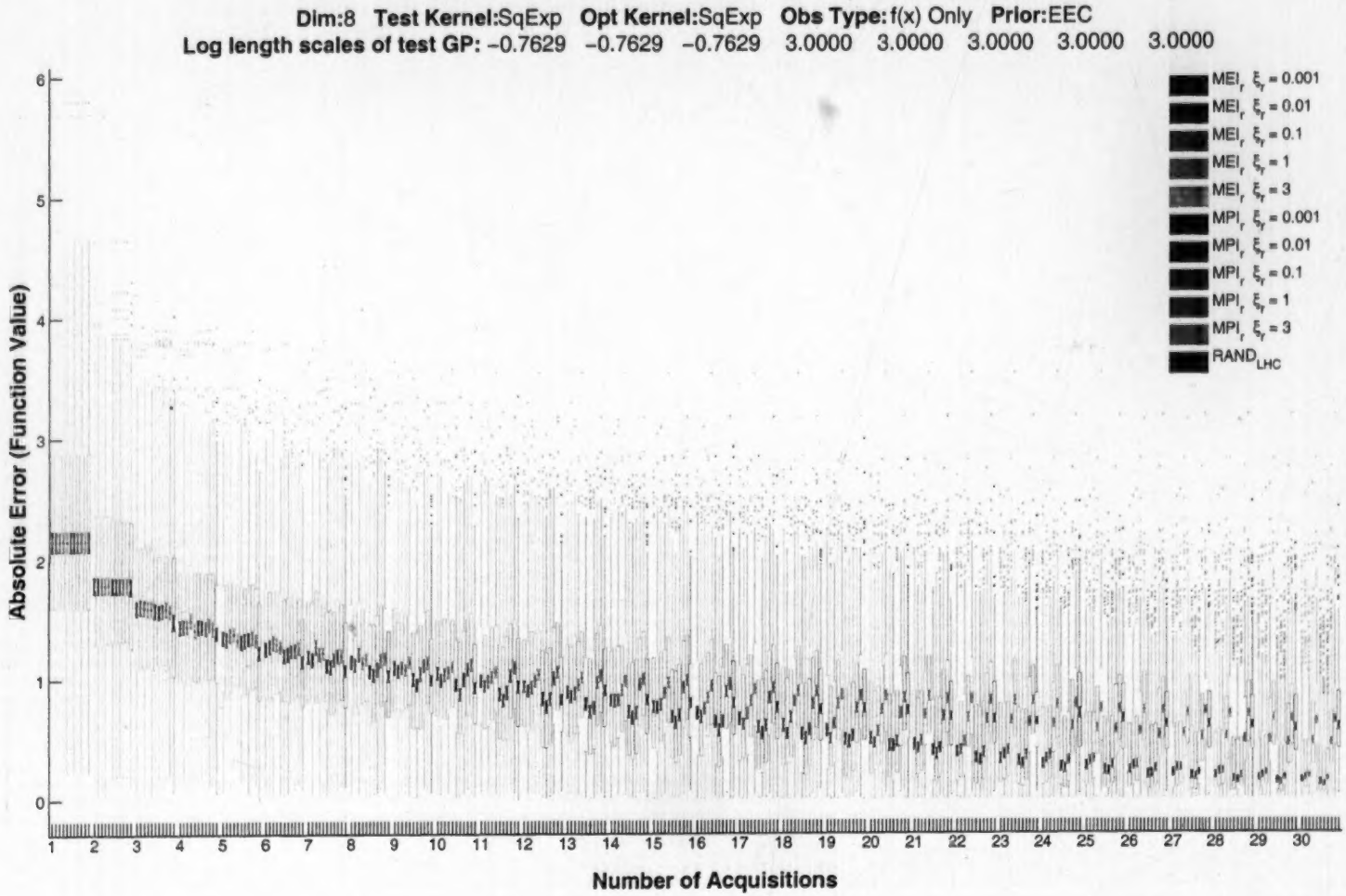
An example posterior mean:

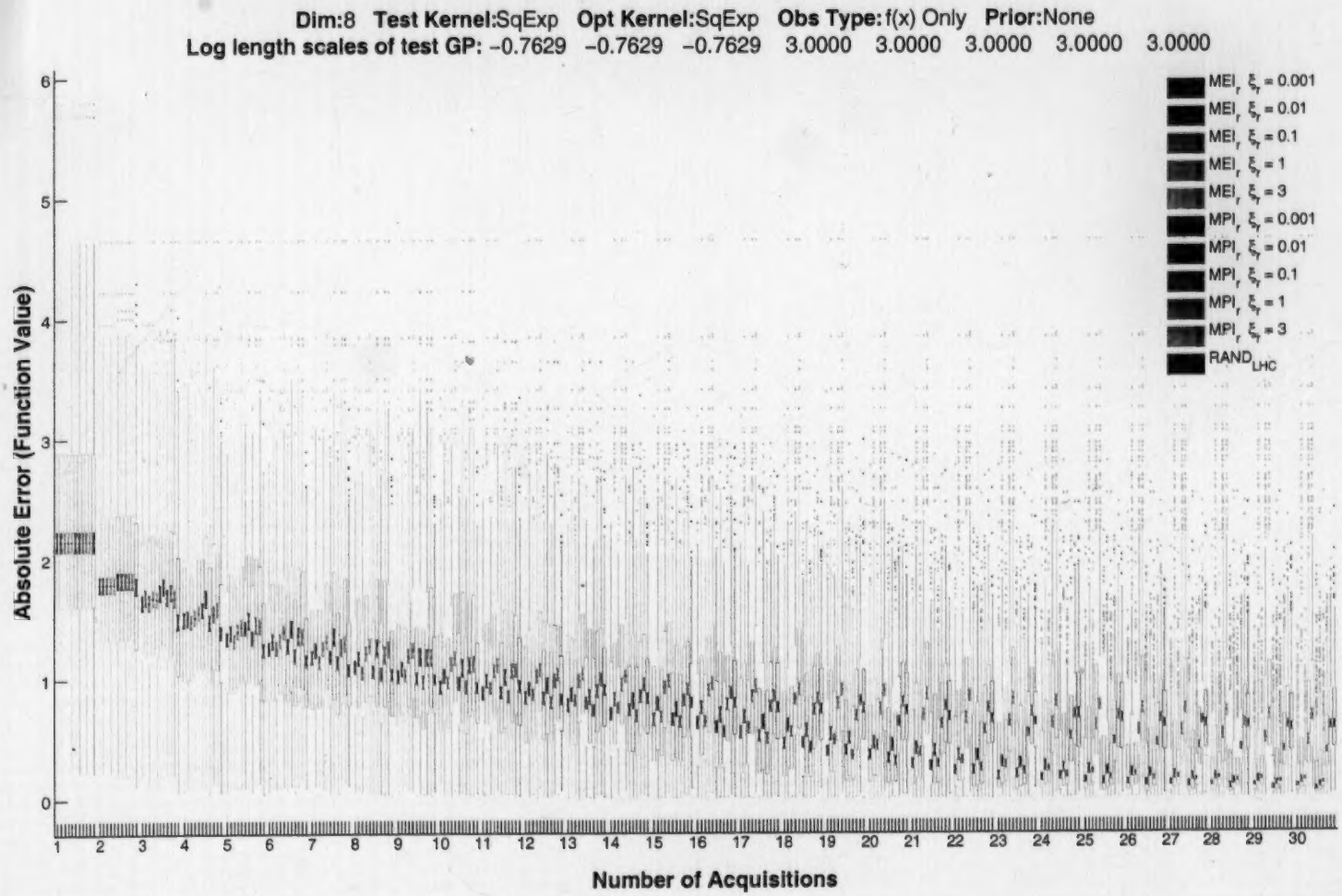


8D squared exponential kernel. First three length scales are short, and the remainder are long. The plot of the example posterior mean is a slice through the origin along dimensions 1 and 2. Only observed function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:8 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: -0.7629 -0.7629 -0.7629 3.0000 3.0000 3.0000 3.0000 3.0000
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:8 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) Only Prior:ILN
Log length scales of test GP: -0.7629 -0.7629 -0.7629 3.0000 3.0000 3.0000 3.0000 3.0000
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

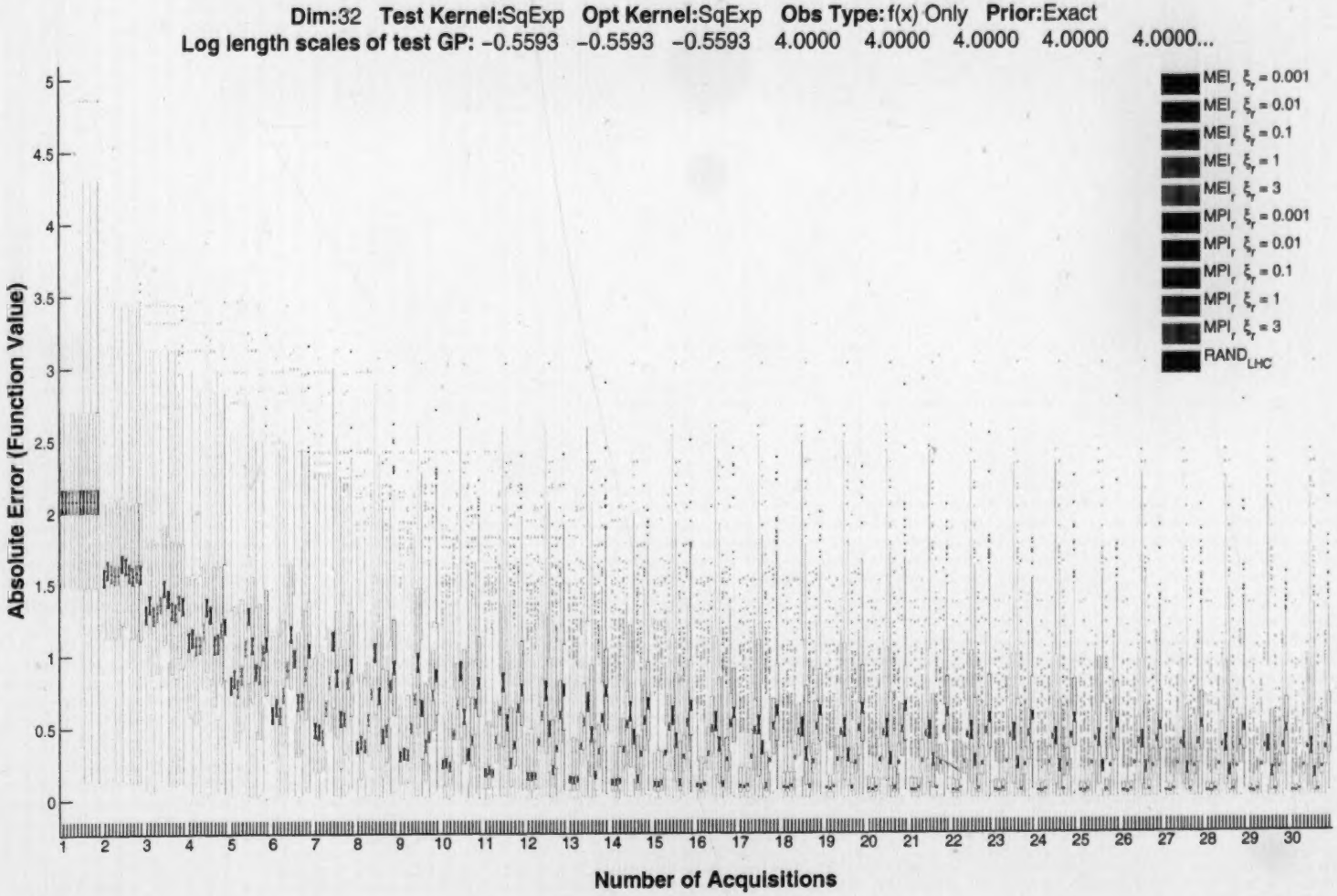

Dim:8 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:EEC
Log length scales of test GP: -0.7629 -0.7629 -0.7629 3.0000 3.0000 3.0000 3.0000 3.0000
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

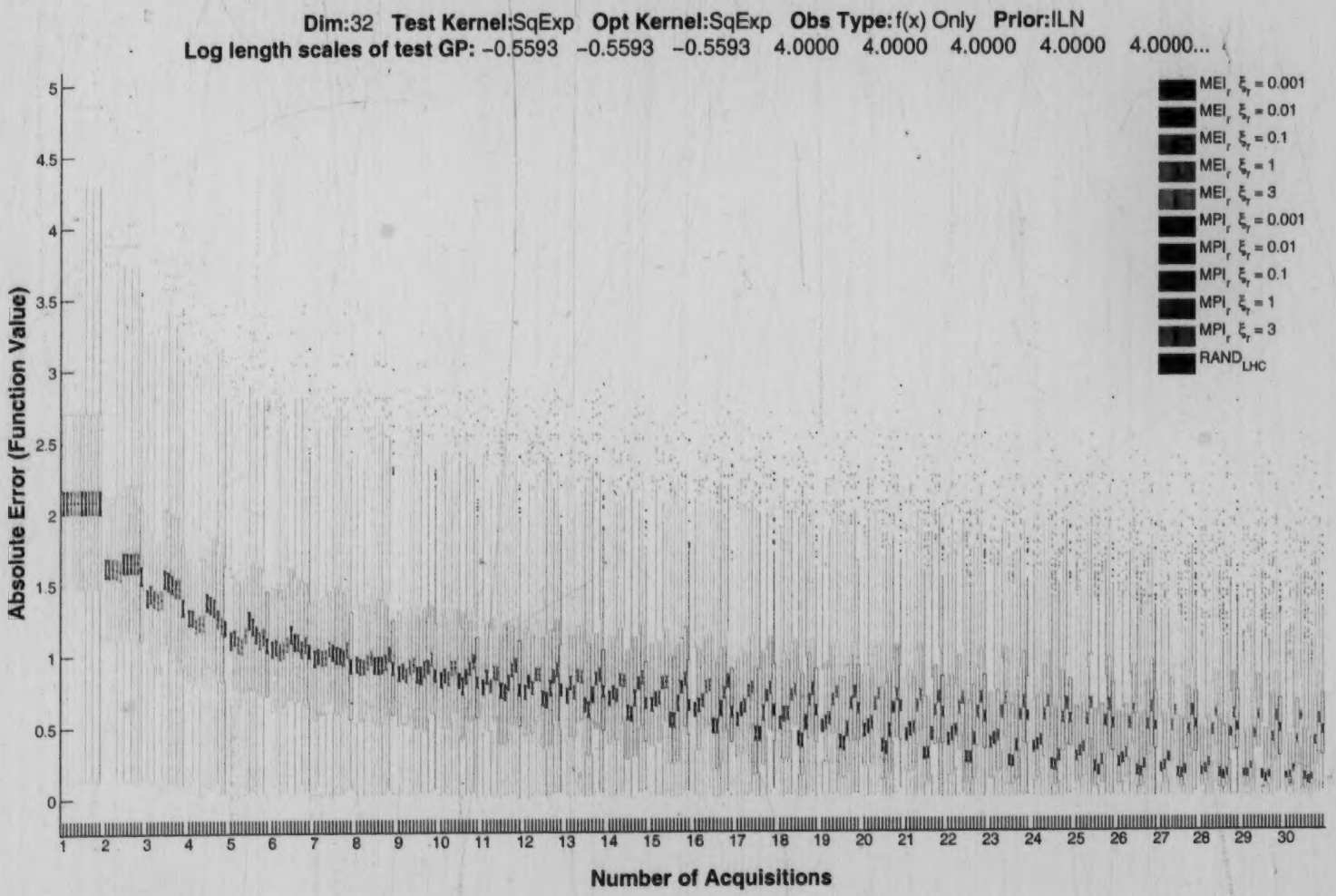

Dim:8 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:None
Log length scales of test GP: -0.7629 -0.7629 -0.7629 3.0000 3.0000 3.0000 3.0000 3.0000
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Absolute Error (Function Value)
0
1
2
3
4
5
6
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

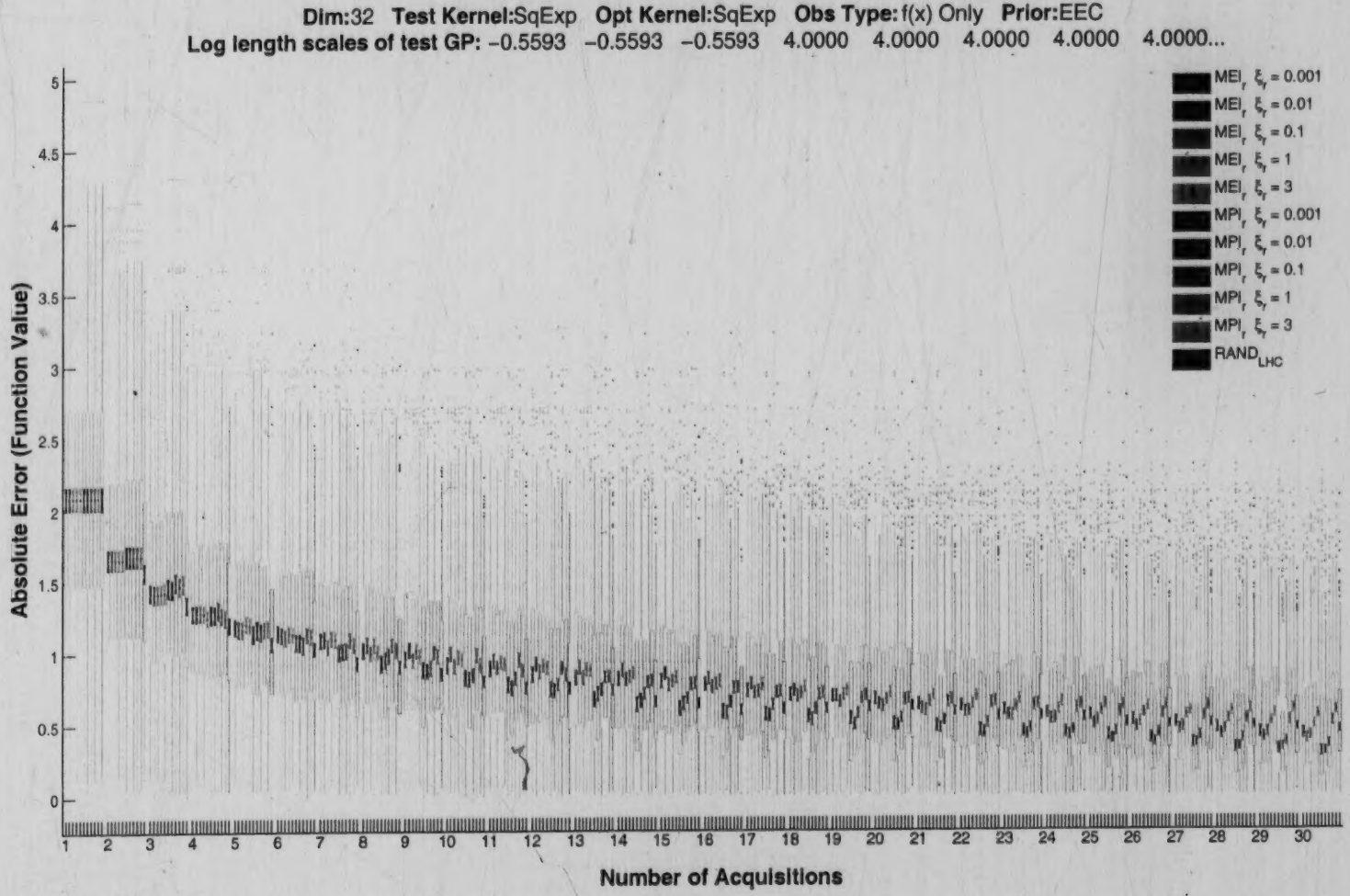
A.6 Results using 32D squared exponential test kernel, unequal length scales

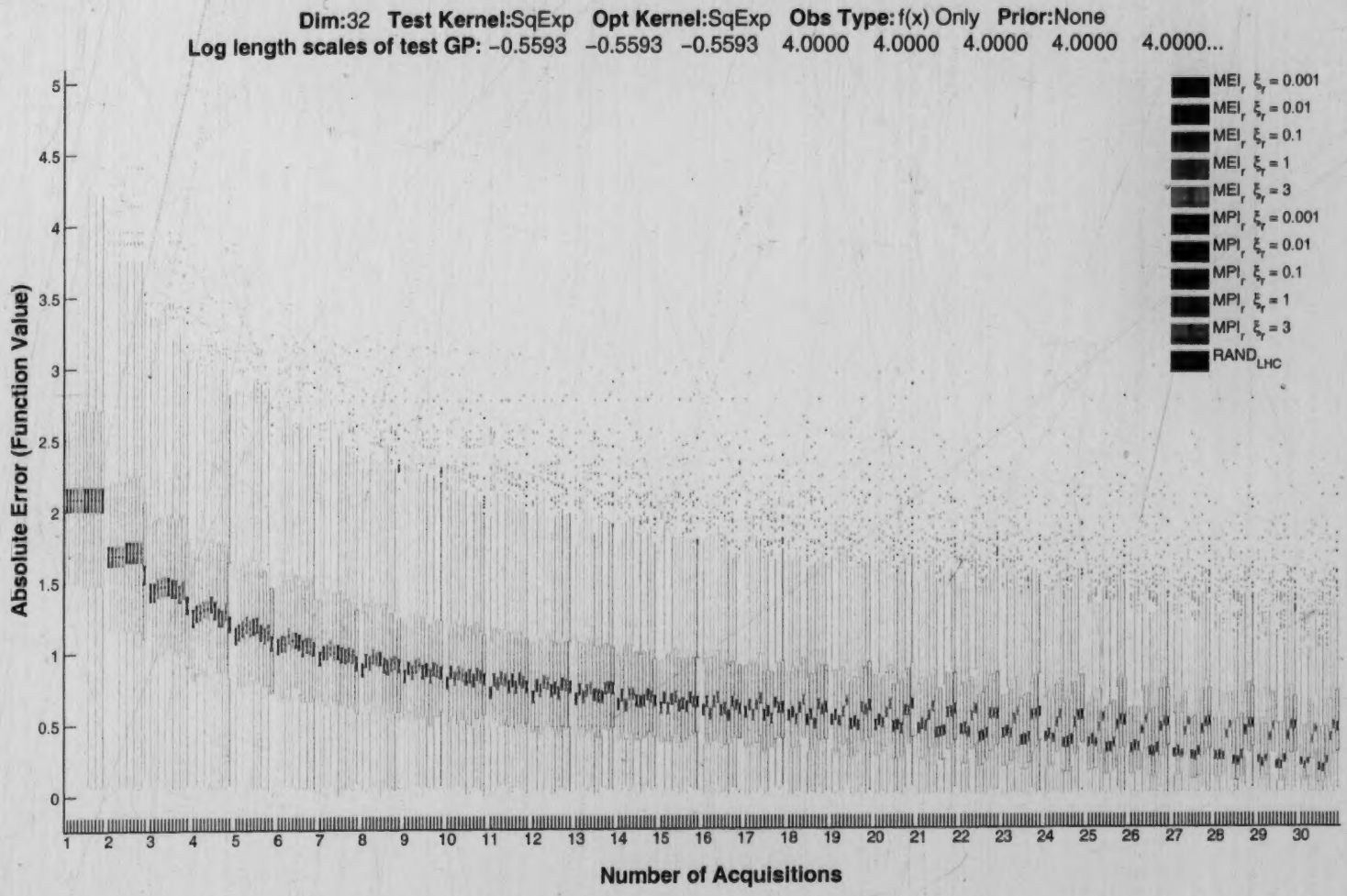
An example posterior mean:

32D squared exponential kernel. First three length scales are short, and the remainder are long. The plot of the example posterior mean is a slice through the origin along dimensions 1 and 2. Only function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:32 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: -0.5593 -0.5593 -0.5593 4.0000 4.0000 4.0000 4.0000 4.0000...
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

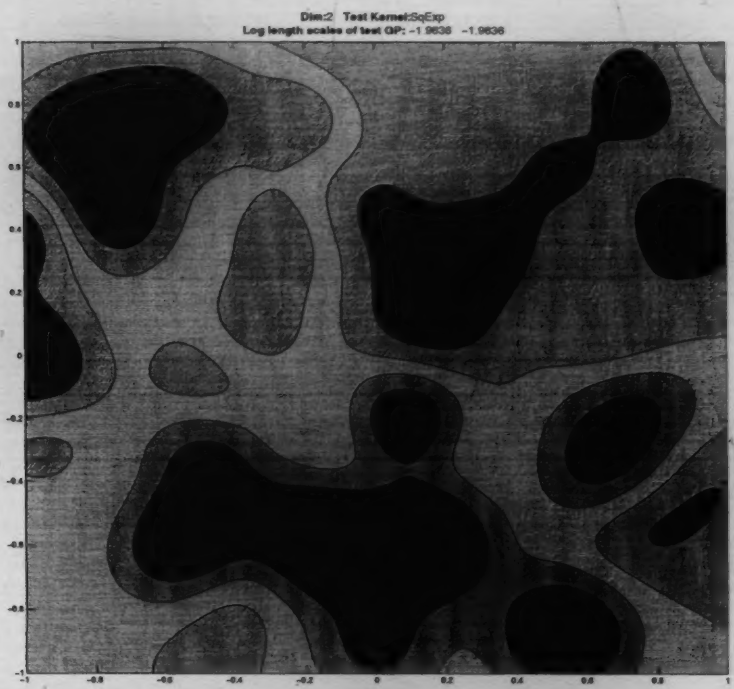

Dim:32 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) Only Prior:ILN
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000 4.0000 4.0000 4.0000...
MEI, ξT = 0.001
MEI, ξT = 0.01
MEI, ξT = 0.1
MEI, ξT = 1
MEI, ξT = 3
MPI, ξT = 0.001
MPI, ξT = 0.01
MPI, ξT = 0.1
MPI, ξT = 1
MPI, ξT = 3
RANDLHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:32 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:EEC
Log length scales of test GP: -0.5593 -0.5593 -0.5593 4.0000 4.0000 4.0000 4.0000 4.0000...
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

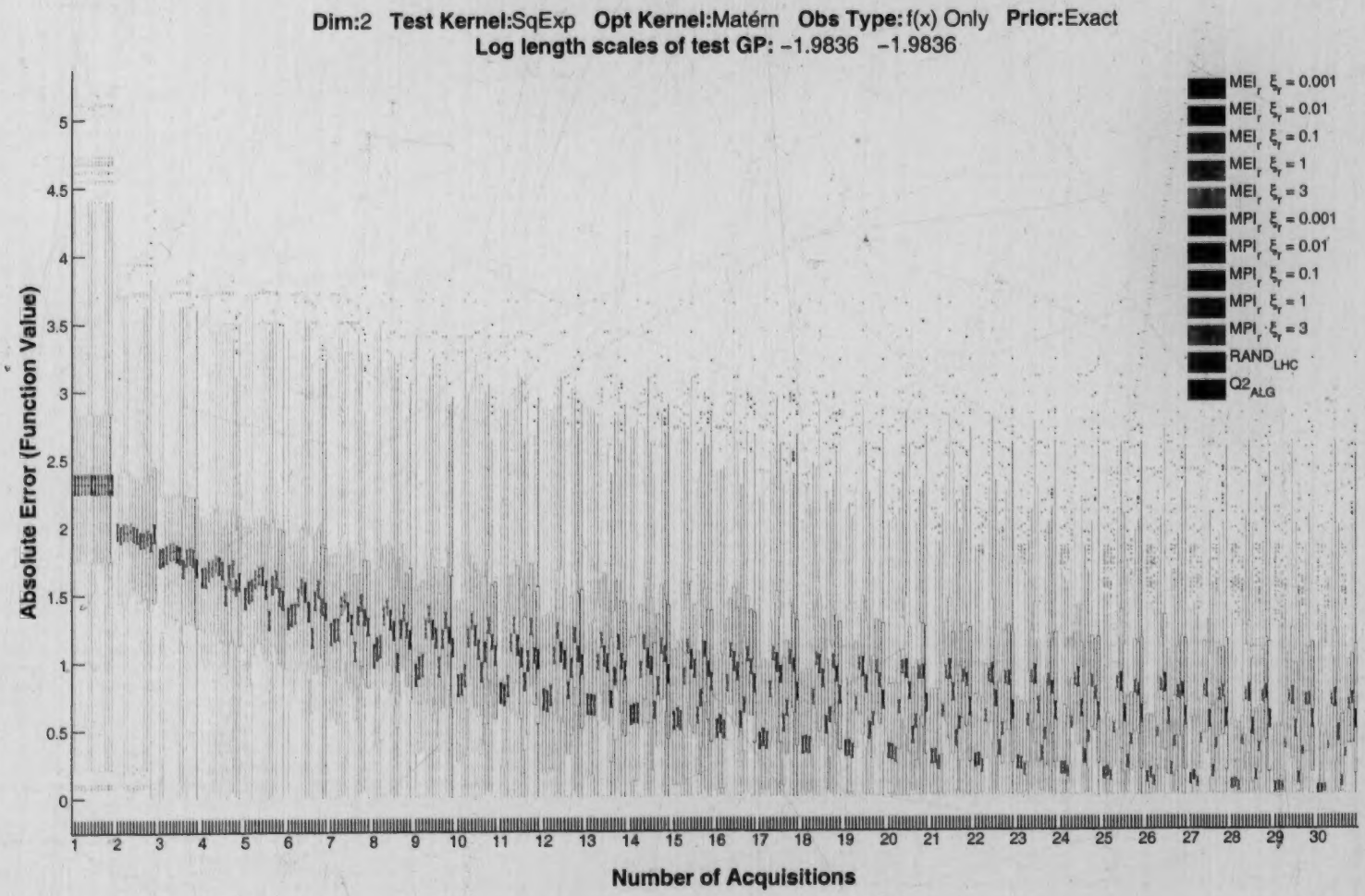

Dim:32 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) Only Prior:None
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000 4.0000 4.0000 4.0000...
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

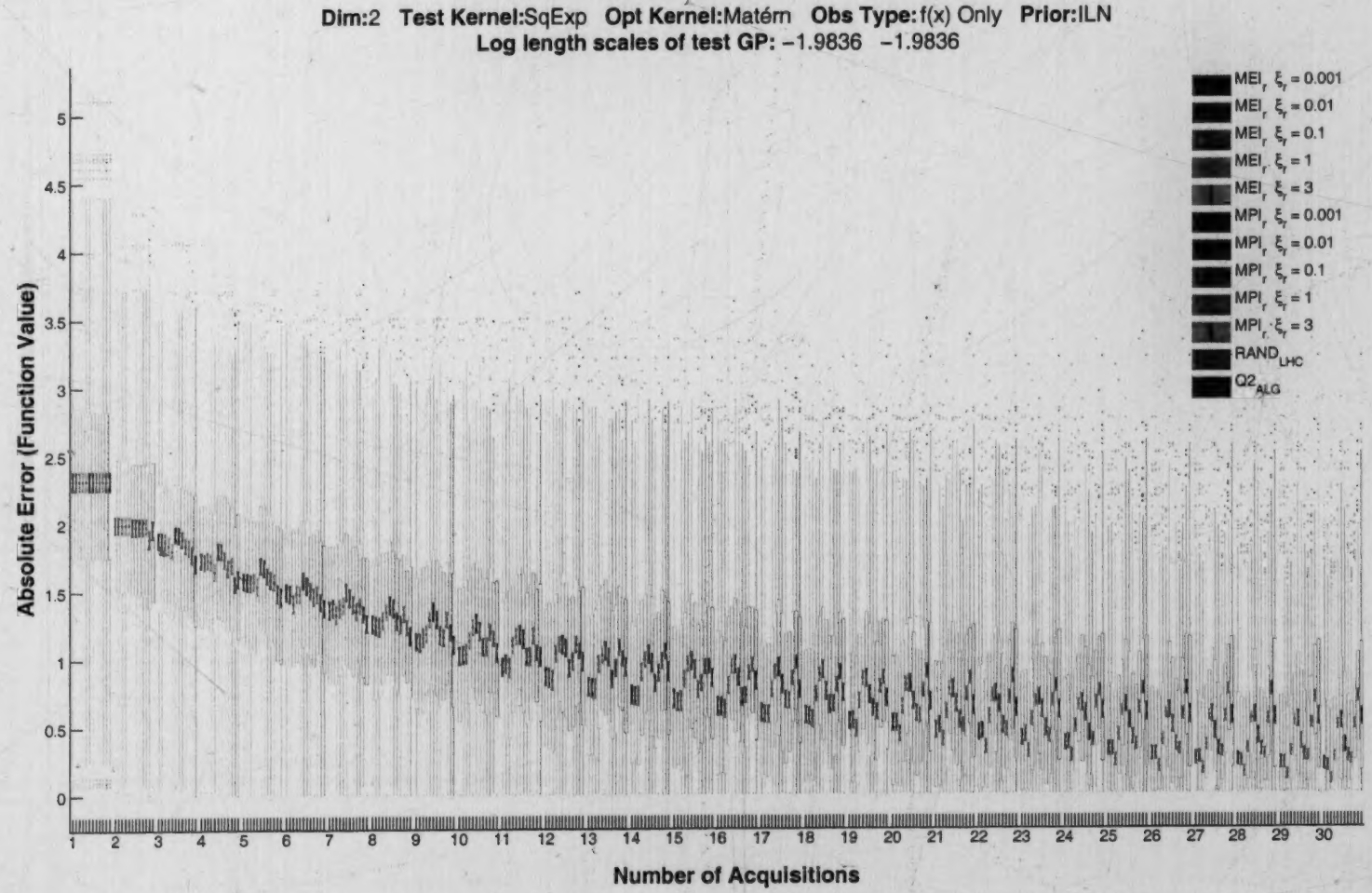
A.7 Results using 2D squared exponential test kernel, equal length scales, Matérn optimization model kernel

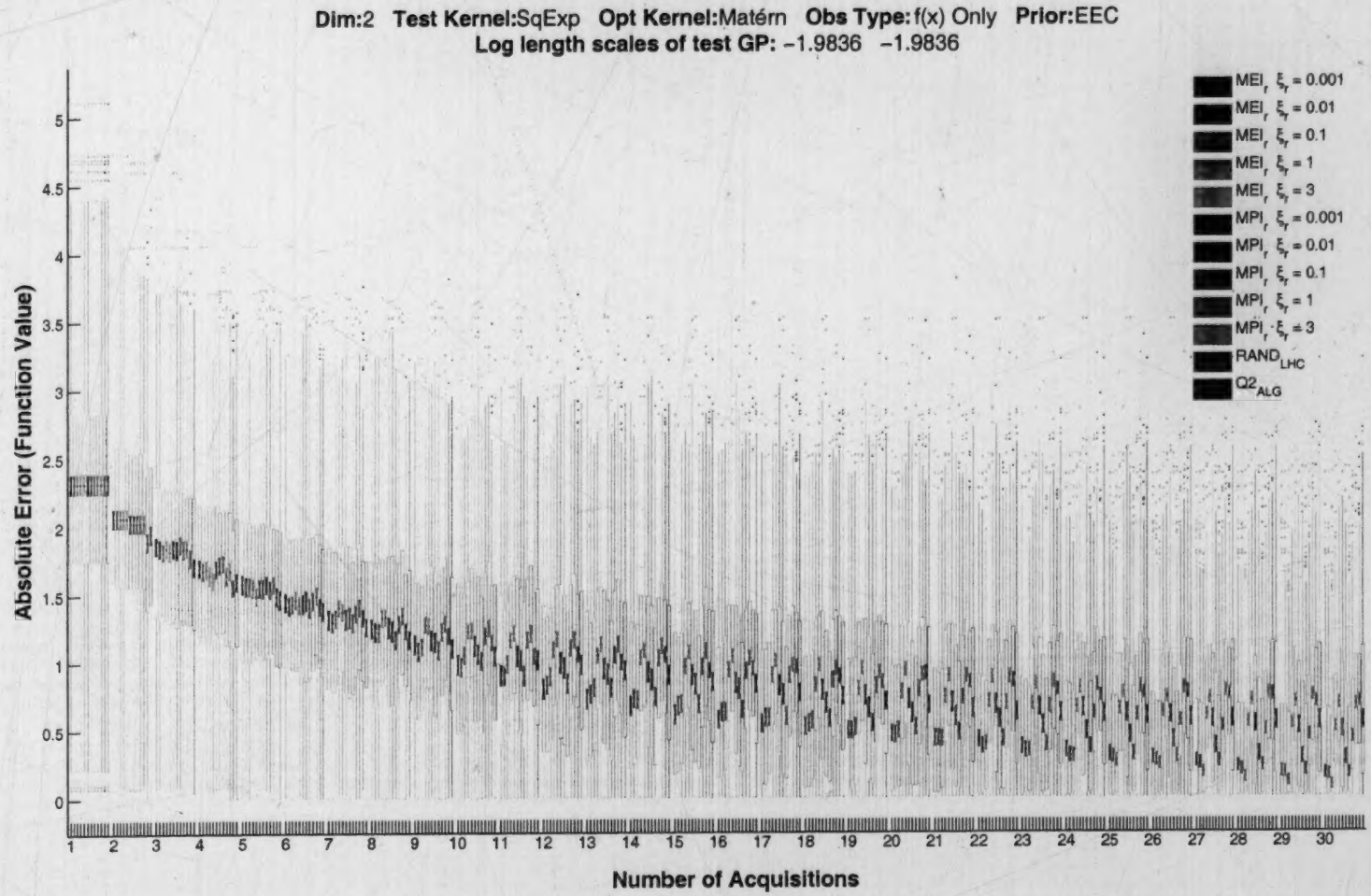
An example posterior mean:

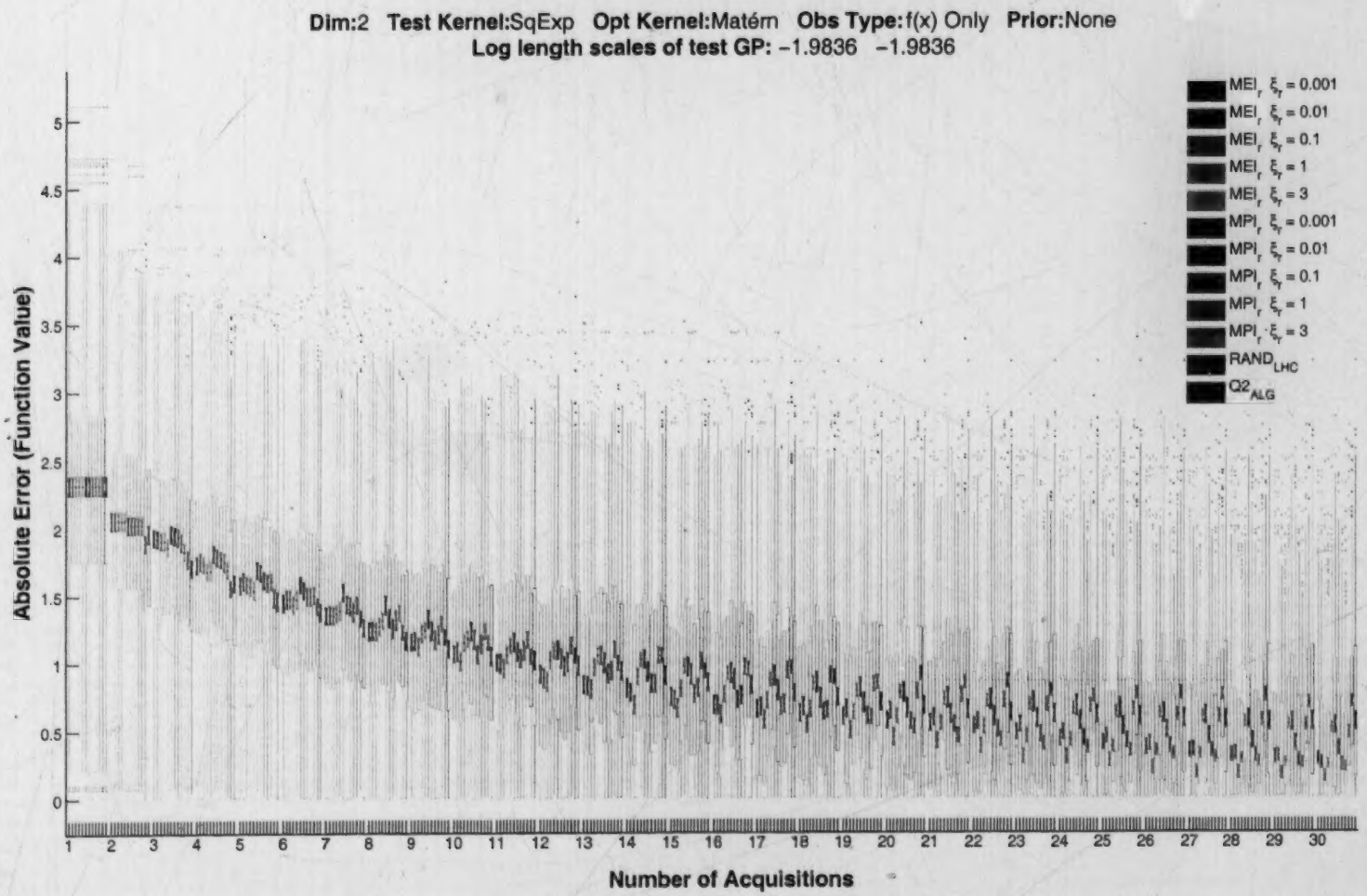


2D squared exponential test kernel with equal length scales. The optimization model instead uses a Matérn kernel. Only function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:SqExp Opt Kernel:Matérn Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: -1.9836 -1.9836
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

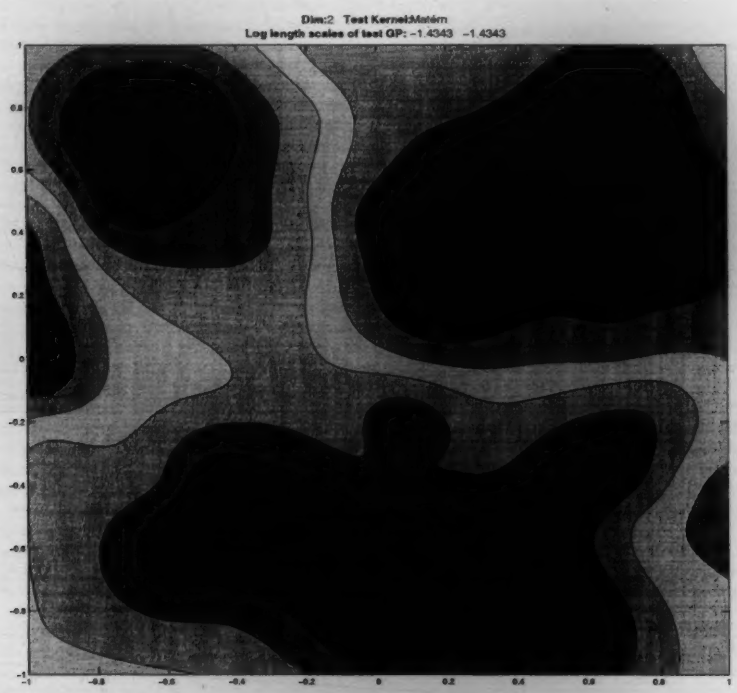

Dim:2 Test Kernel:SqExp Opt Kernel:Matérn Obs Type:f(x) Only Prior:ILN
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND_LHC
Q2_ALG


Dim:2 Test Kernel:SqExp Opt Kernel:Matérn Obs Type: f(x) Only Prior:EEC
Log length scales of test GP: −1.9836 −1.9836
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

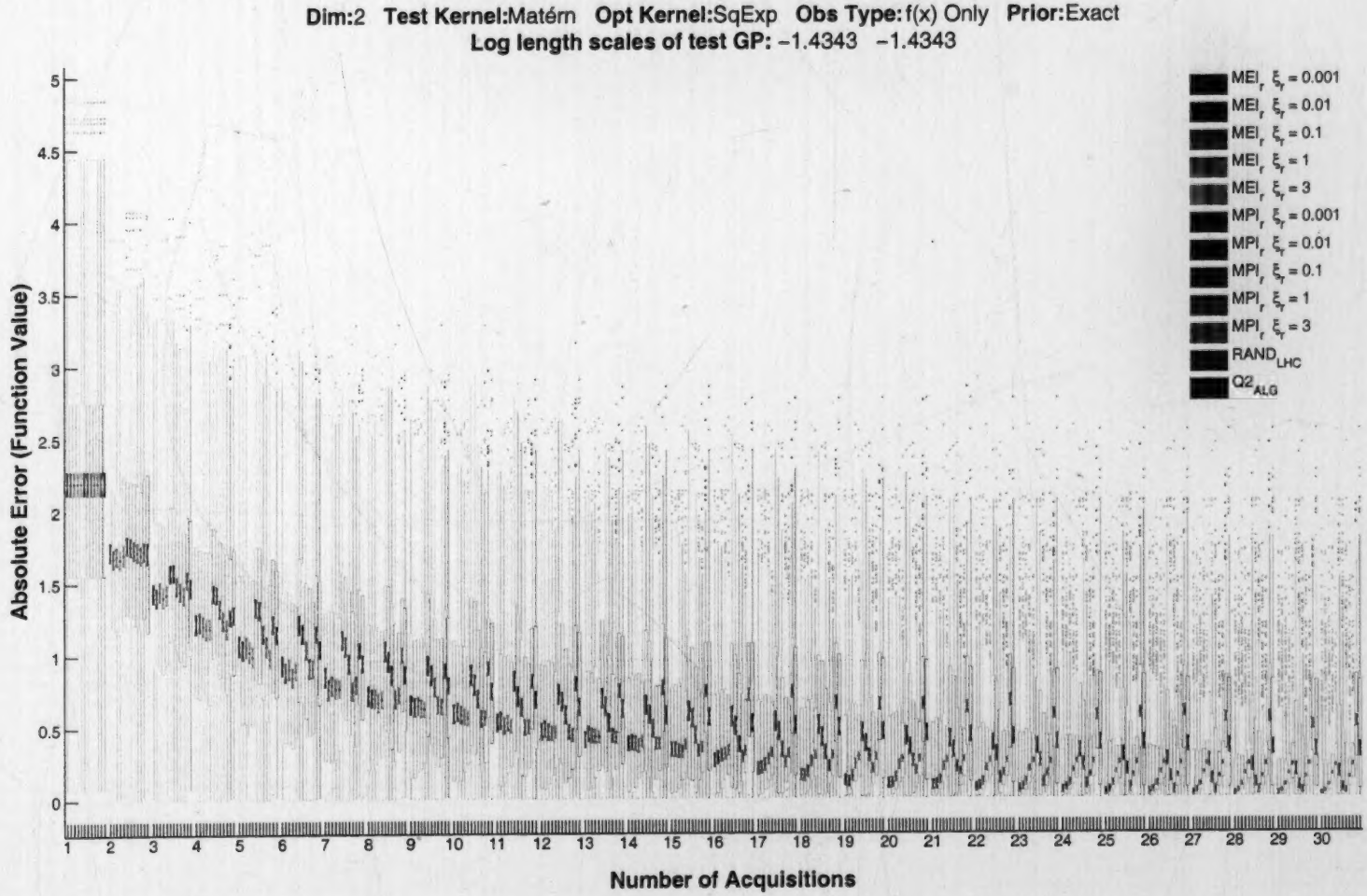

Dim:2 Test Kernel:SqExp Opt Kernel:Matérn Obs Type:f(x) Only Prior:None
Log length scales of test GP: -1.9836 -1.9836
Absolute Error (Function Value)
Number of Acquisitions
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG

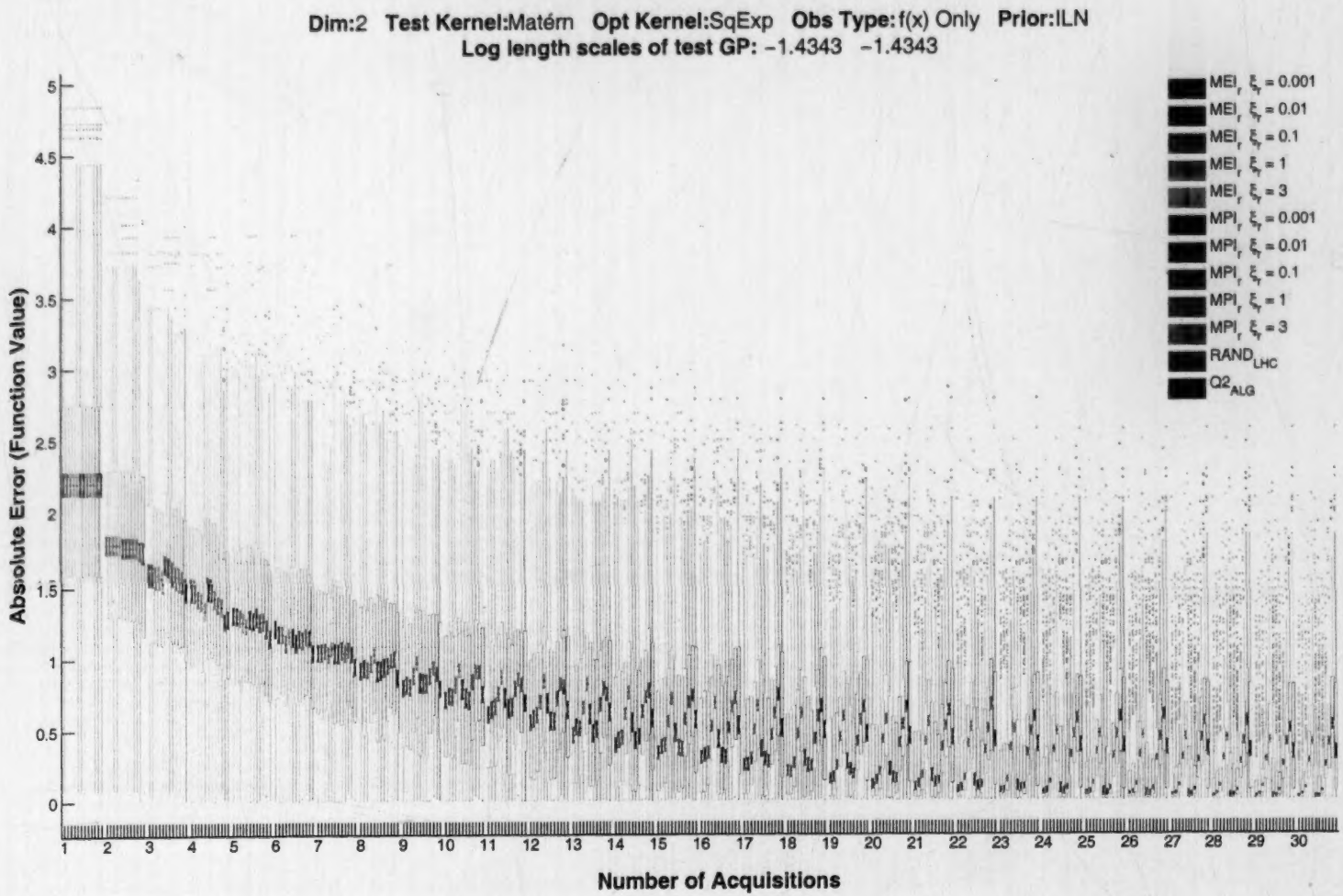
A.8 Results using 2D Matérn test kernel, equal length scales, squared exponential optimization model kernel

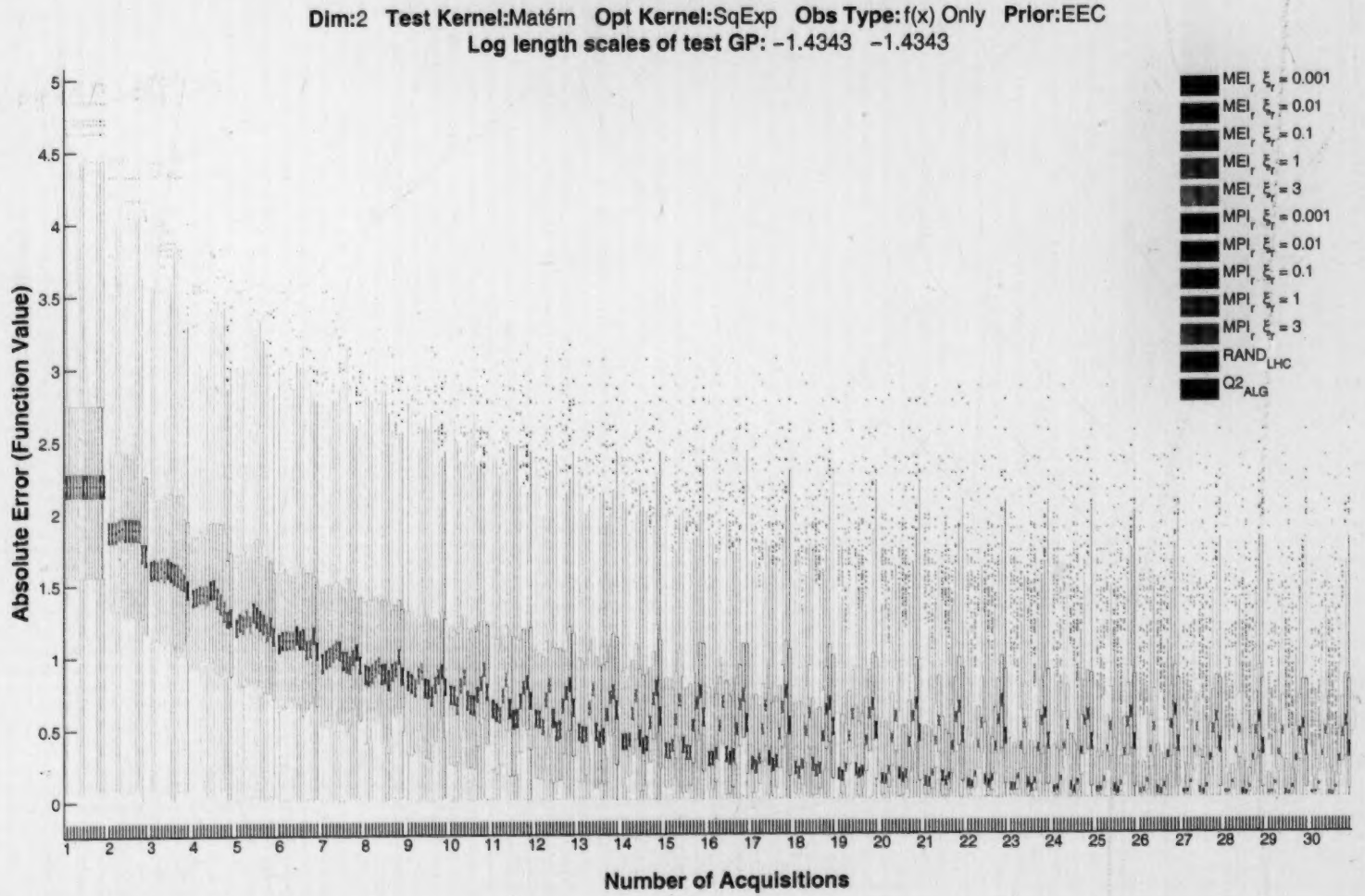
An example posterior mean:

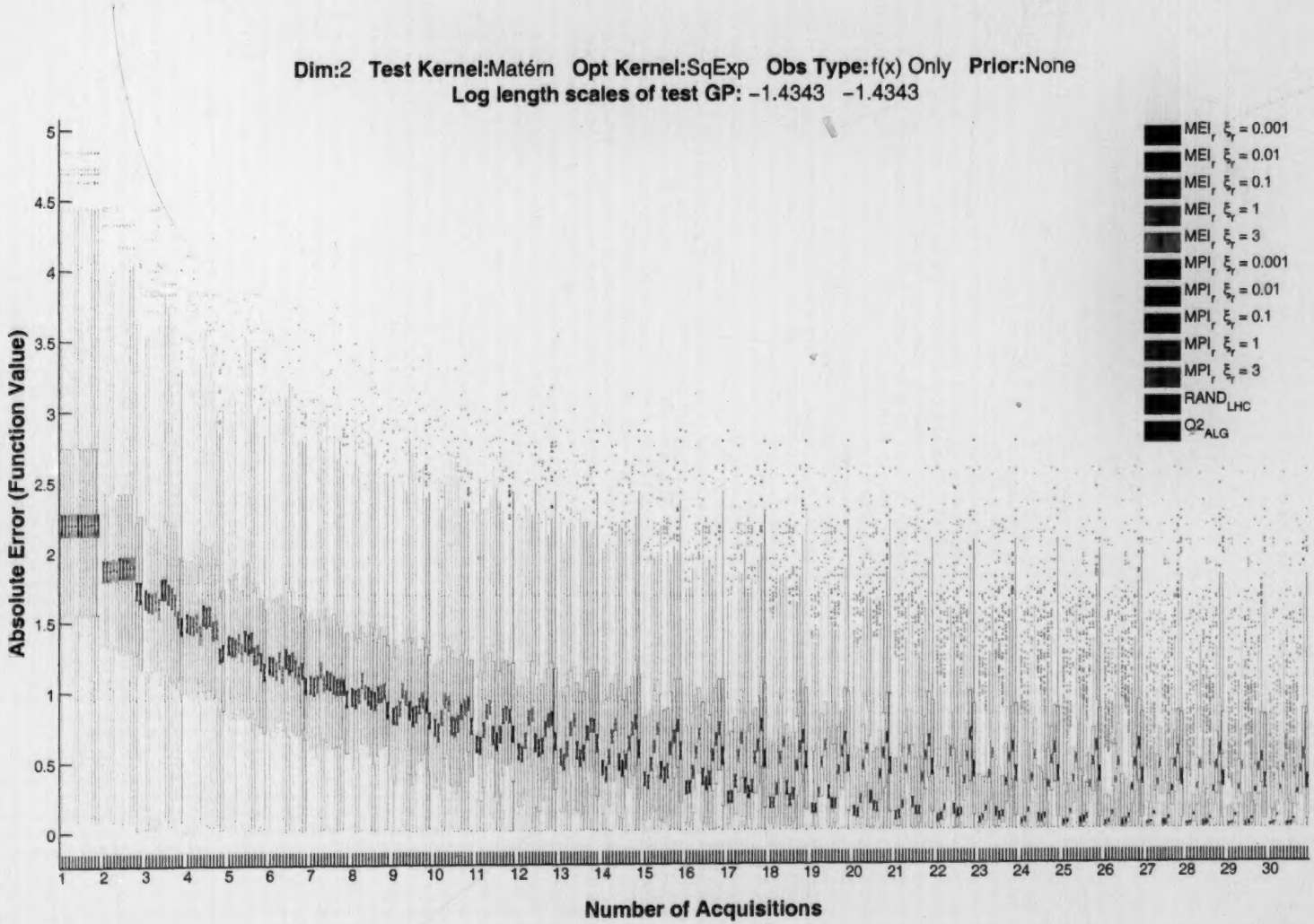


2D Matérn test kernel with equal length scales. The optimization model instead uses a squared exponential. Only observed function values are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:Matérn Opt Kernel:SqExp Obs Type: f(x) Only Prior:Exact
Log length scales of test GP: −1.4343 −1.4343
MEI, ξ_r = 0.001
MEI, ξ_r = 0.01
MEI, ξ_r = 0.1
MEI, ξ_r = 1
MEI, ξ_r = 3
MPI, ξ_r = 0.001
MPI, ξ_r = 0.01
MPI, ξ_r = 0.1
MPI, ξ_r = 1
MPI, ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

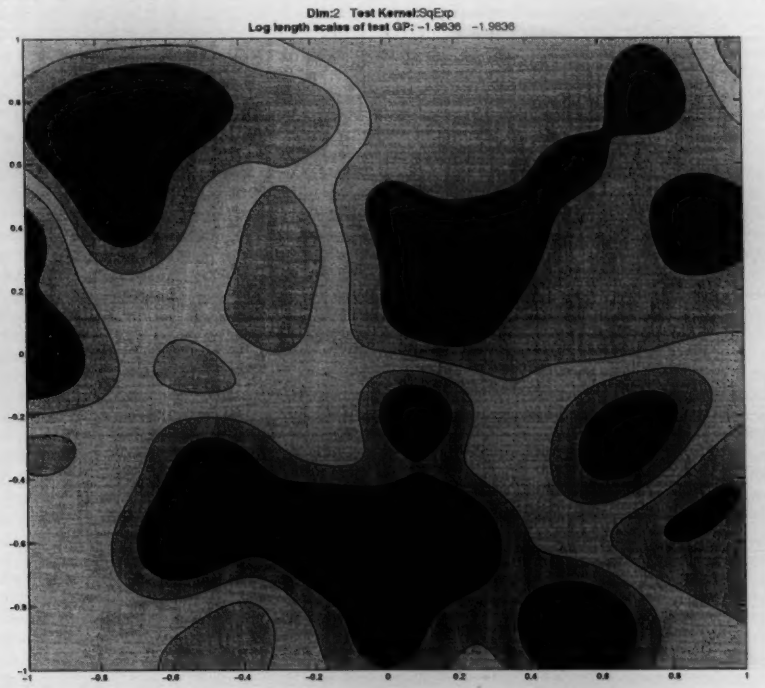

Dim:2 Test Kernel:Matérn Opt Kernel:SqExp Obs Type: f(x) Only Prior:ILN
Log length scales of test GP: −1.4343 −1.4343
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Q2_ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:2 Test Kernel:Matérn Opt Kernel:SqExp Obs Type:f(x) Only Prior:EEC
Log length scales of test GP: -1.4343 -1.4343
MEI, ξ = 0.001
MEI, ξ = 0.01
MEI, ξ = 0.1
MEI, ξ = 1
MEI, ξ = 3
MPI, ξ = 0.001
MPI, ξ = 0.01
MPI, ξ = 0.1
MPI, ξ = 1
MPI, ξ = 3
RAND LHC
Q2 ALG
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

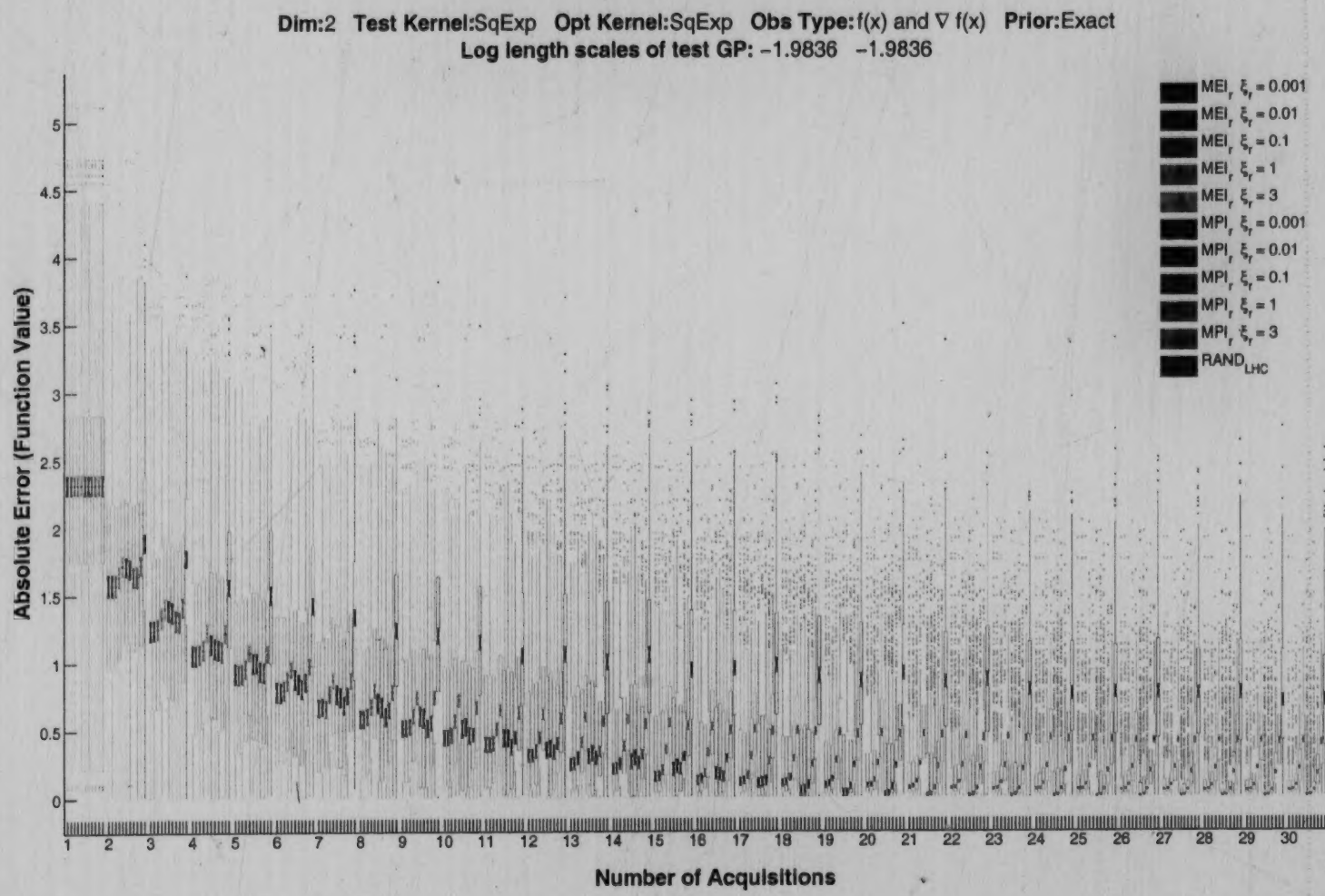

Dim:2 Test Kernel:Matérn Opt Kernel:SqExp Obs Type: f(x) Only Prior:None
Log length scales of test GP: −1.4343 −1.4343
MEI, ξr = 0.001
MEI, ξr = 0.01
MEI, ξr = 0.1
MEI, ξr = 1
MEI, ξr = 3
MPI, ξr = 0.001
MPI, ξr = 0.01
MPI, ξr = 0.1
MPI, ξr = 1
MPI, ξr = 3
RAND LHC
Q2 ALG
Absolute Error (Function Value)
Number of Acquisitions

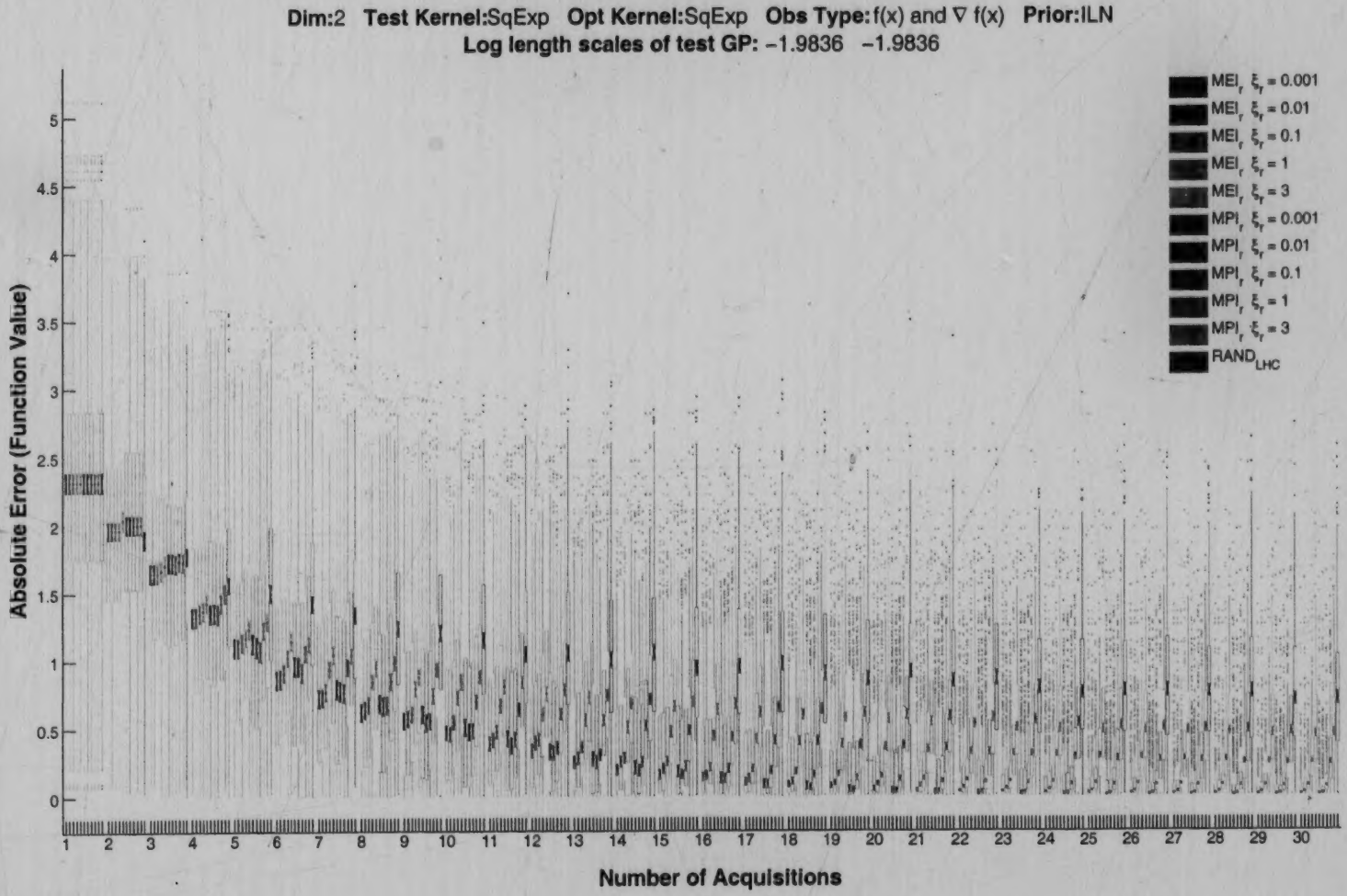
A.9 Results using 2D squared exponential test kernel, equal length scales, gradients included

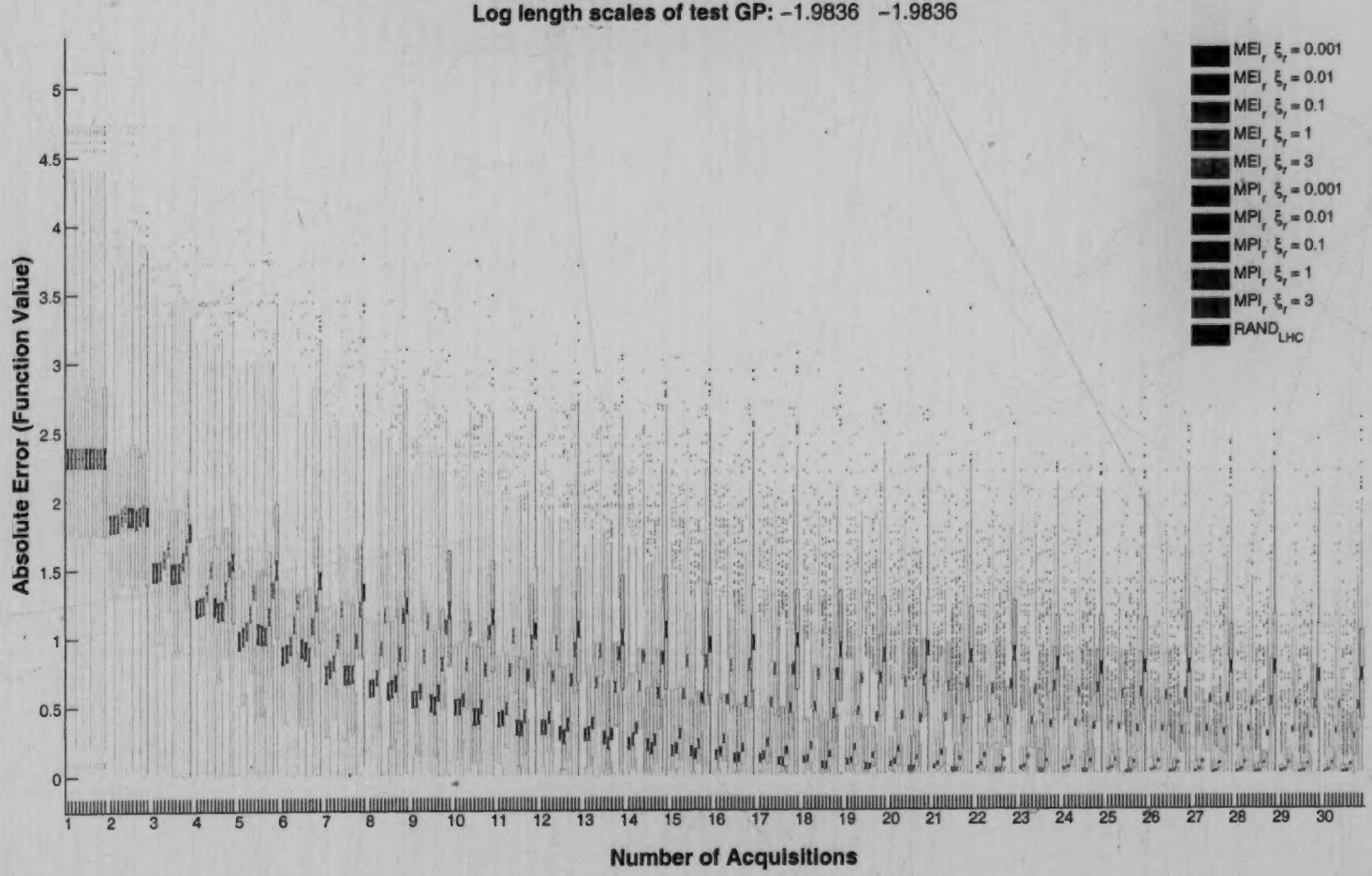
An example posterior mean:

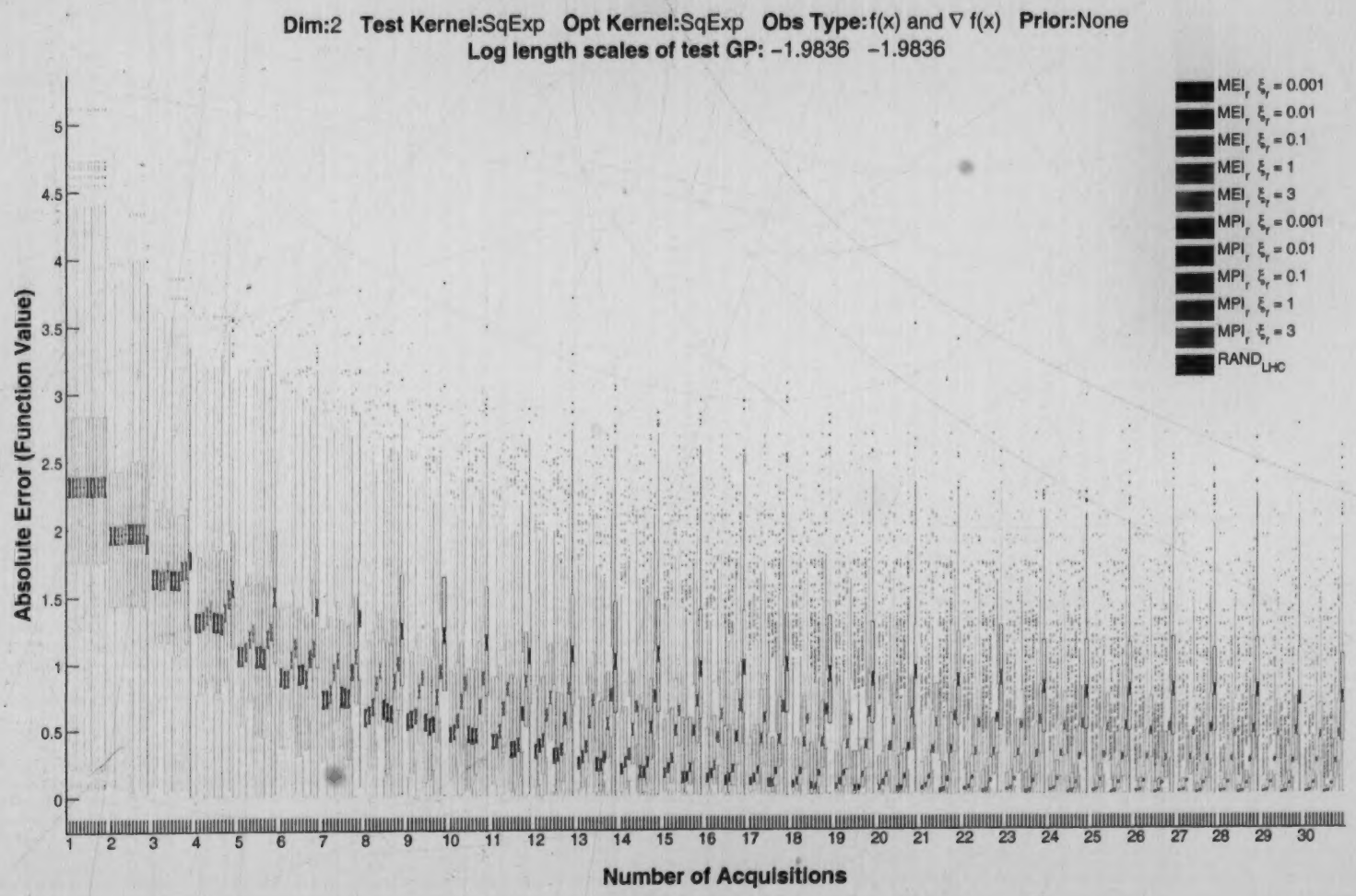


2D squared exponential test kernel with equal length scales. The optimization model uses a squared exponential kernel. Observed function values and gradients are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) and ∇ f(x) Prior:Exact
Log length scales of test GP: -1.9836 -1.9836
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

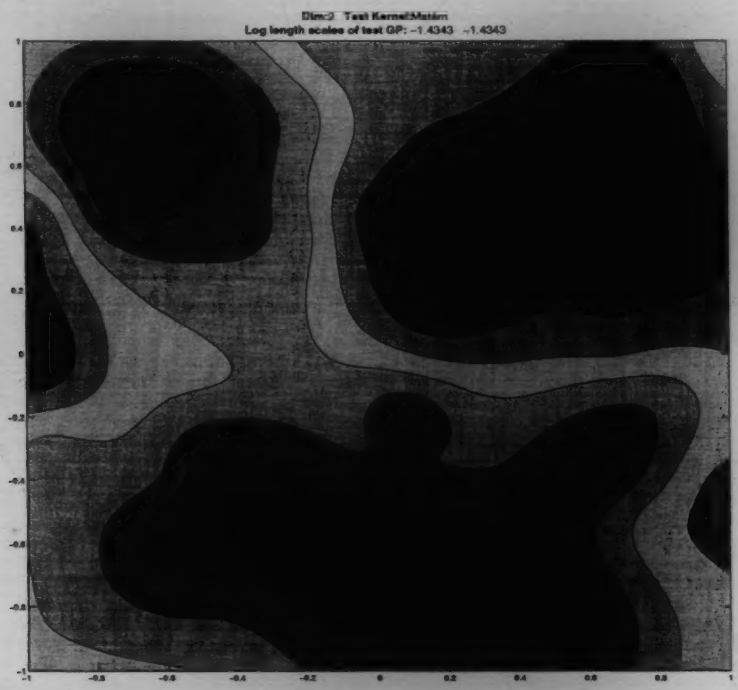

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) and ∇ f(x) Prior:ILN
Log length scales of test GP: −1.9836 −1.9836
MEI, ξ_r = 0.001
MEI, ξ_r = 0.01
MEI, ξ_r = 0.1
MEI, ξ_r = 1
MEI, ξ_r = 3
MPI, ξ_r = 0.001
MPI, ξ_r = 0.01
MPI, ξ_r = 0.1
MPI, ξ_r = 1
MPI, ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type: f(x) and ∇ f(x) Prior:EEC
Log length scales of test GP: −1.9836 −1.9836
MEI, ξr = 0.001
MEI, ξr = 0.01
MEI, ξr = 0.1
MEI, ξr = 1
MEI, ξr = 3
MPI, ξr = 0.001
MPI, ξr = 0.01
MPI, ξr = 0.1
MPI, ξr = 1
MPI, ξr = 3
RAND LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

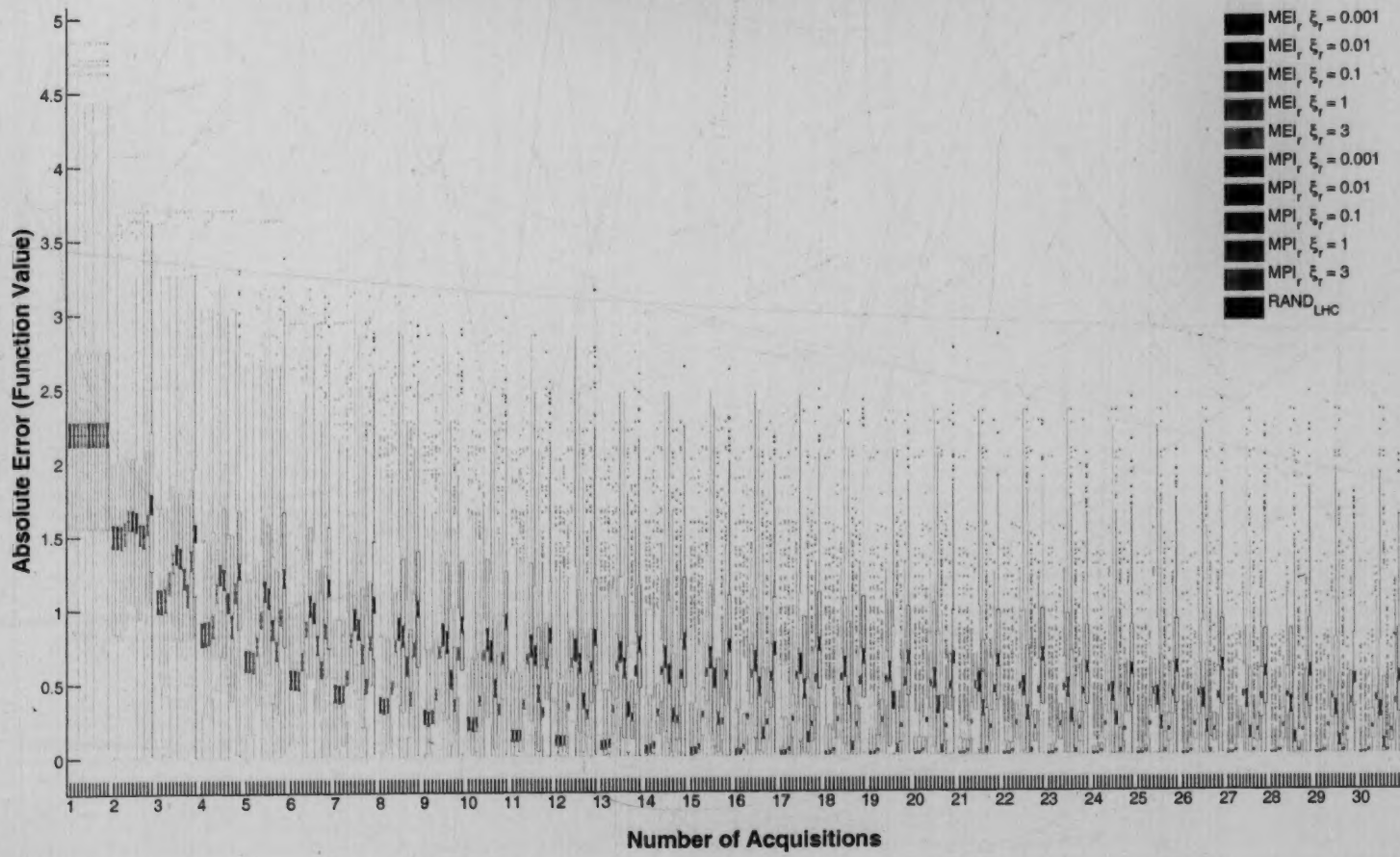

Dim:2 Test Kernel:SqExp Opt Kernel:SqExp Obs Type:f(x) and ∇ f(x) Prior:None
Log length scales of test GP: −1.9836 −1.9836
MEI, ξ_r = 0.001
MEI, ξ_r = 0.01
MEI, ξ_r = 0.1
MEI, ξ_r = 1
MEI, ξ_r = 3
MPI, ξ_r = 0.001
MPI, ξ_r = 0.01
MPI, ξ_r = 0.1
MPI, ξ_r = 1
MPI, ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions

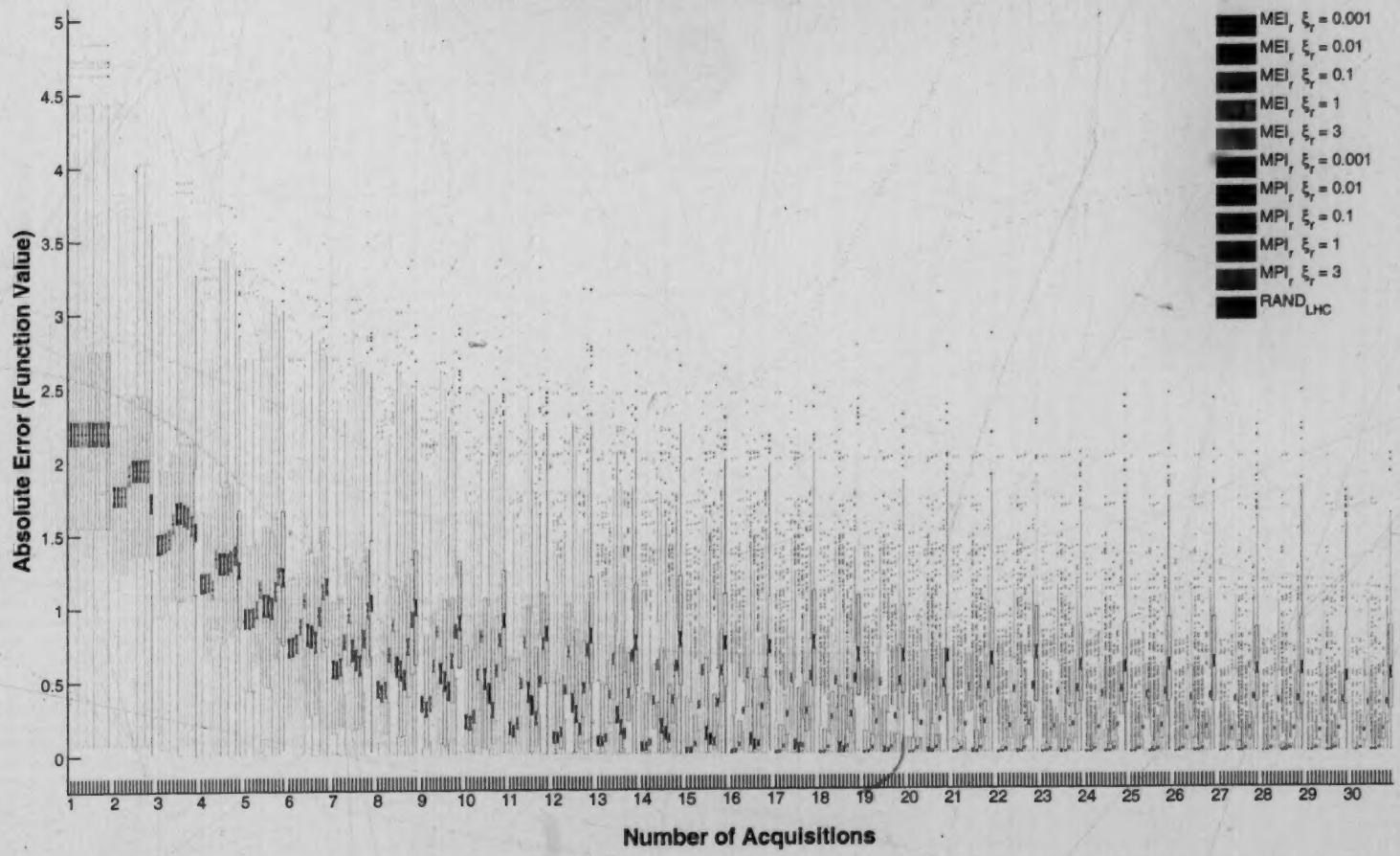
A.10 Results using 2D Matérn test kernel, equal length scales, gradients included

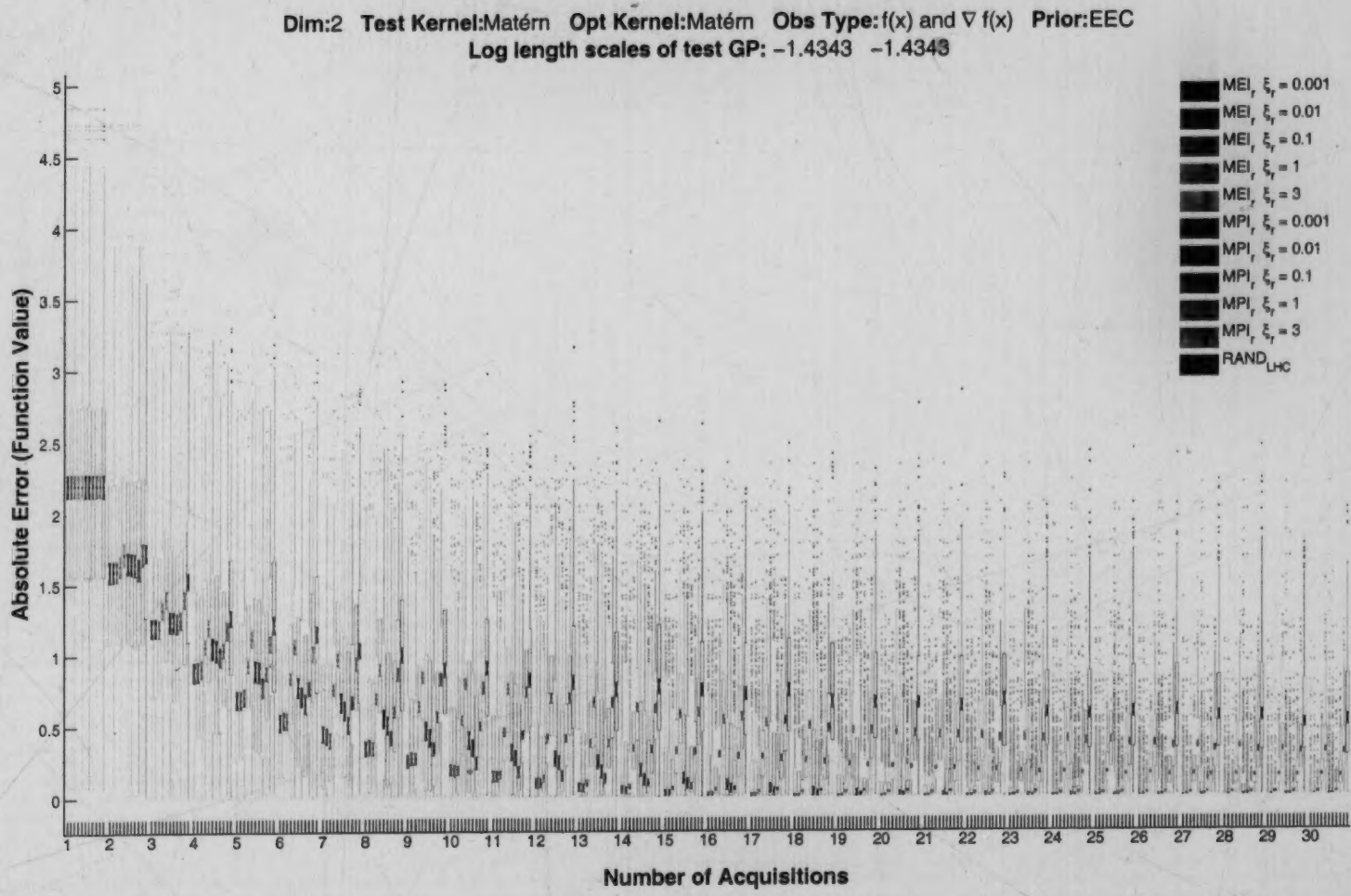
An example posterior mean:

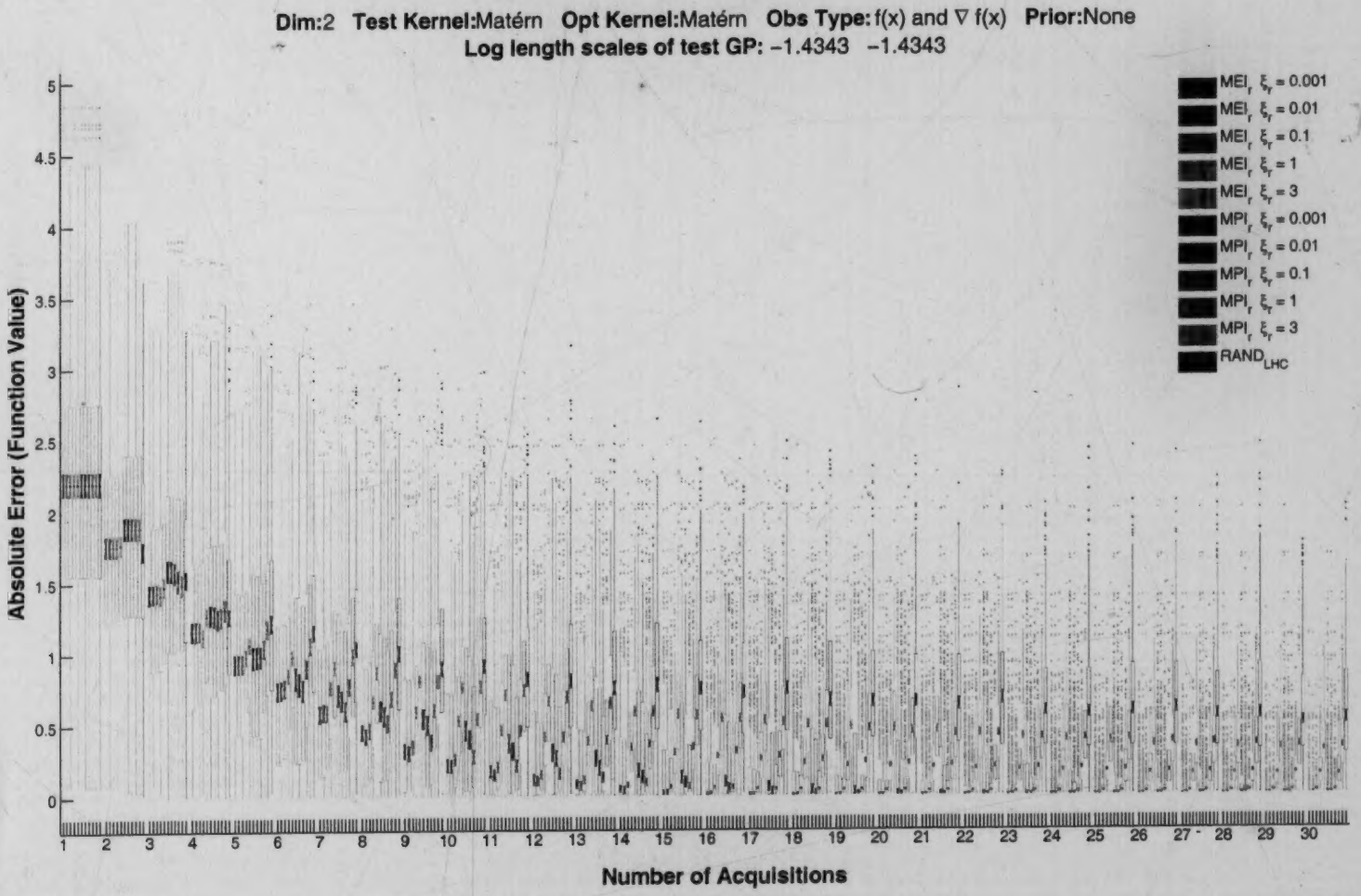


2D Matérn kernel with equal length scales. The optimization model also uses a Matérn kernel. Observed function values and gradients are used in building the optimization model. 500 functions are sampled from the test model.


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type:f(x) and ∇ f(x) Prior:Exact
Log length scales of test GP: −1.4343 −1.4343
Absolute Error (Function Value)
Number of Acquisitions
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) and ∇ f(x) Prior:ILN
Log length scales of test GP: −1.4343 −1.4343
MEI, ξ_Y = 0.001
MEI, ξ_Y = 0.01
MEI, ξ_Y = 0.1
MEI, ξ_Y = 1
MEI, ξ_Y = 3
MPI, ξ_Y = 0.001
MPI, ξ_Y = 0.01
MPI, ξ_Y = 0.1
MPI, ξ_Y = 1
MPI, ξ_Y = 3
RAND_LHC
Absolute Error (Function Value)
Number of Acquisitions


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) and ∇ f(x) Prior:EEC
Log length scales of test GP: −1.4343 −1.4343
MEI, ξ_r = 0.001
MEI, ξ_r = 0.01
MEI, ξ_r = 0.1
MEI, ξ_r = 1
MEI, ξ_r = 3
MPI, ξ_r = 0.001
MPI, ξ_r = 0.01
MPI, ξ_r = 0.1
MPI, ξ_r = 1
MPI, ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Number of Acquisitions


Dim:2 Test Kernel:Matérn Opt Kernel:Matérn Obs Type: f(x) and ∇ f(x) Prior:None
Log length scales of test GP: −1.4343 −1.4343
MEI_r ξ_r = 0.001
MEI_r ξ_r = 0.01
MEI_r ξ_r = 0.1
MEI_r ξ_r = 1
MEI_r ξ_r = 3
MPI_r ξ_r = 0.001
MPI_r ξ_r = 0.01
MPI_r ξ_r = 0.1
MPI_r ξ_r = 1
MPI_r ξ_r = 3
RAND_LHC
Absolute Error (Function Value)
Number of Acquisitions

Appendix B

Effect of Priors on Performance

B.1 With function values only

The graphs in this section illustrate the effect of different ξ_r on performance of MEI_r and MPI_r on each of the test models when learning length scales using different priors. The observation models here were built using observed function values only.

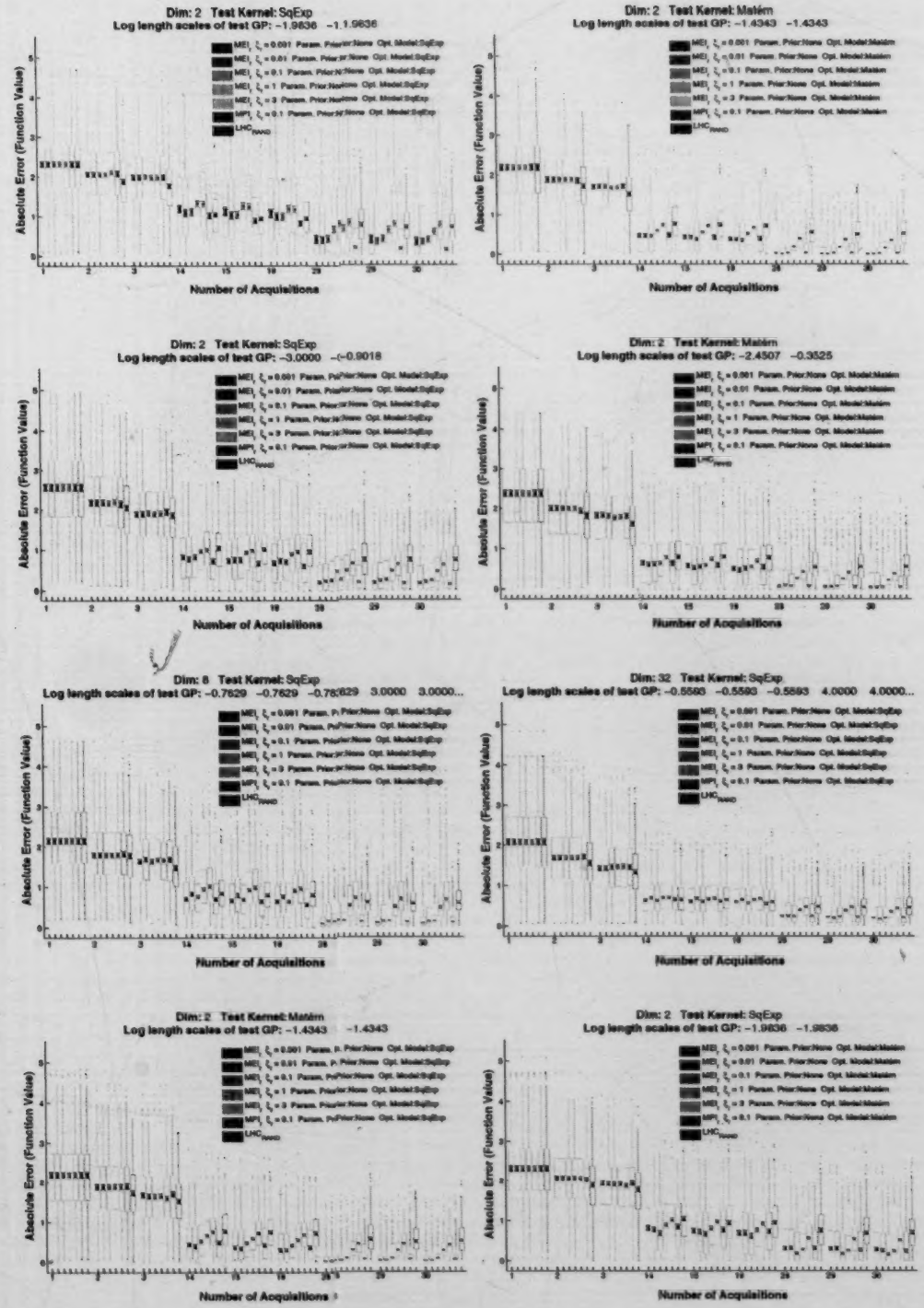

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.1: Performance

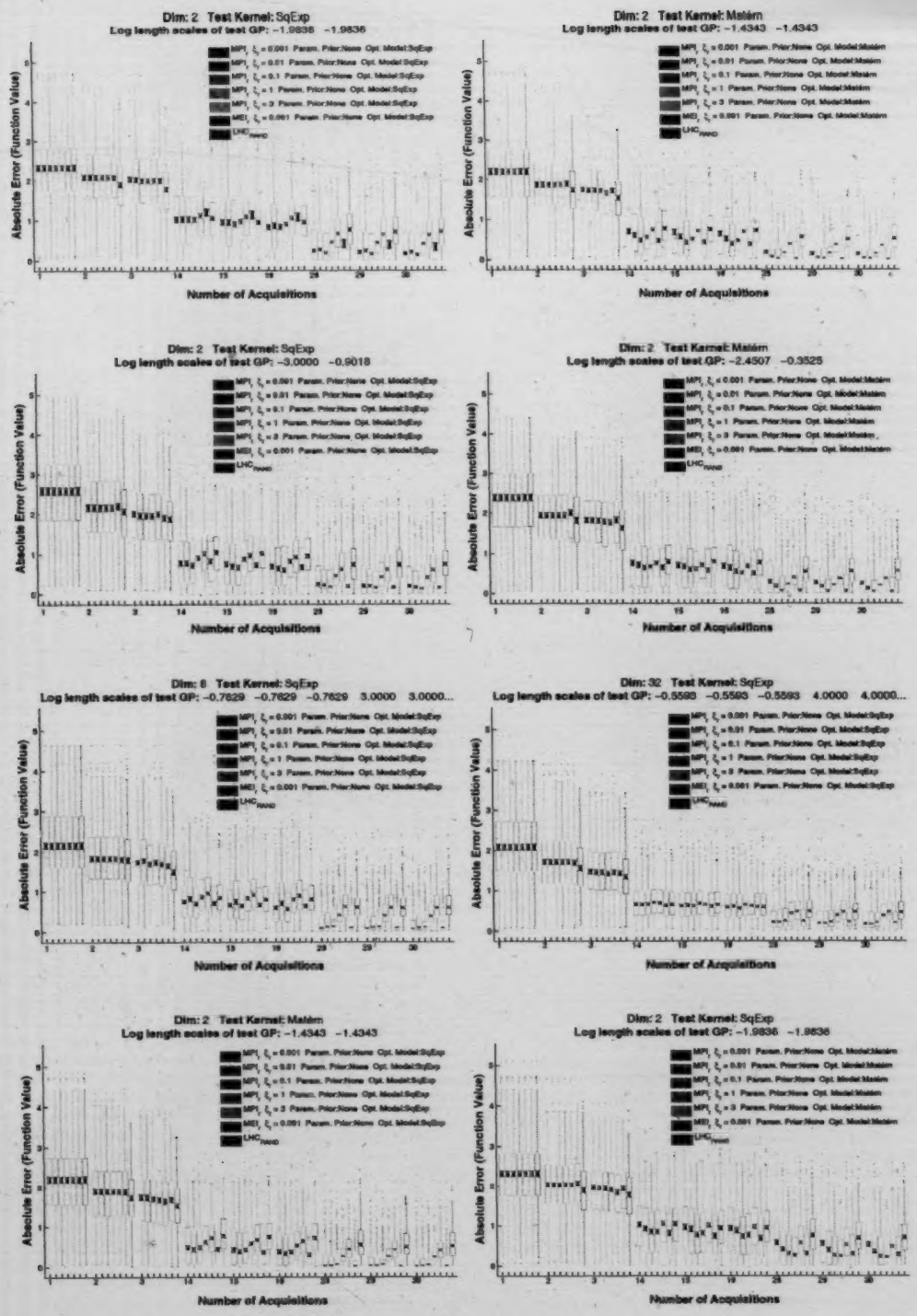

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.2: Performance using no prior

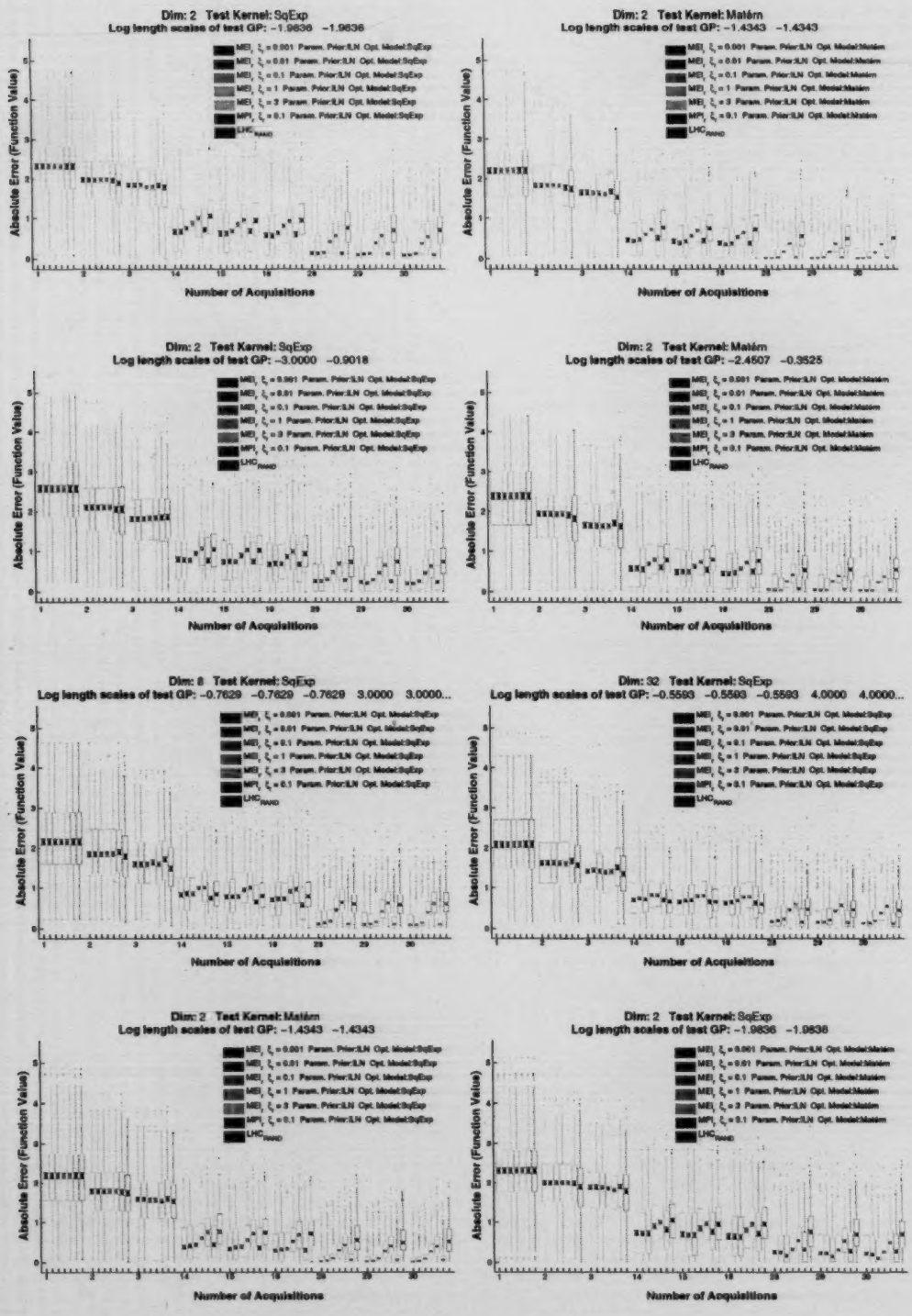

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.3: Performance using ILN prior

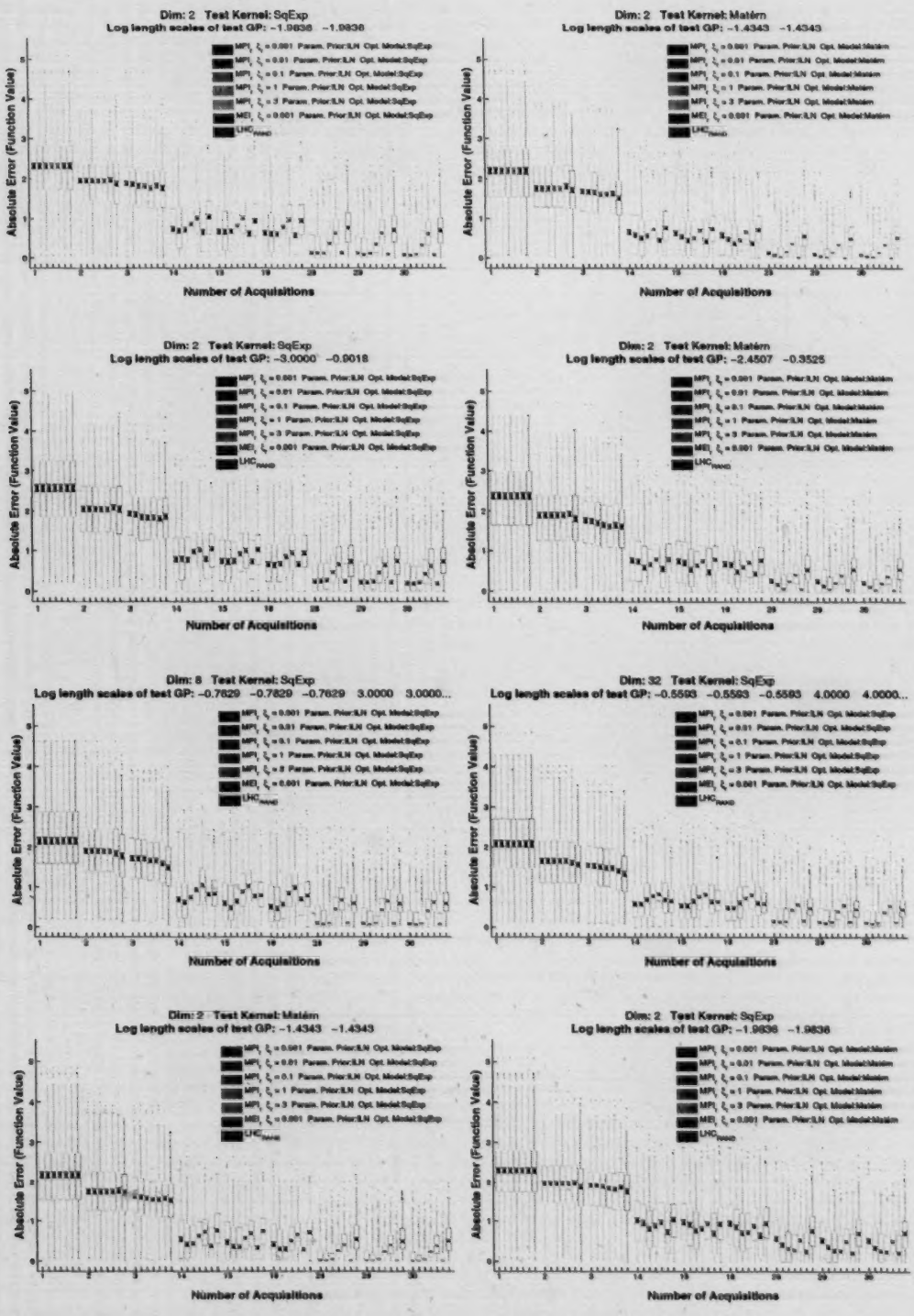

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.0018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.4: Performance using ILN prior

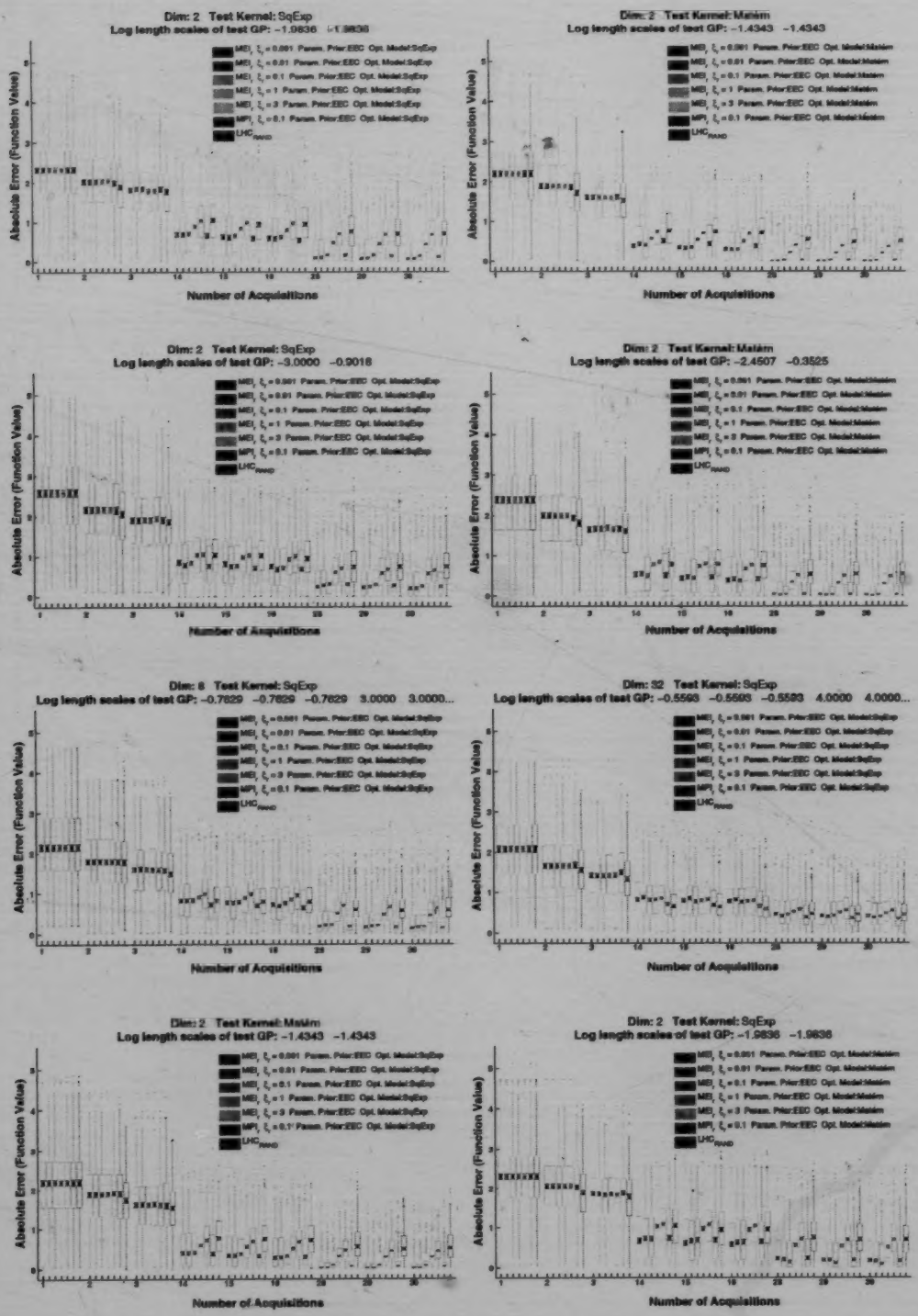

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.5: Performance using EEC prior

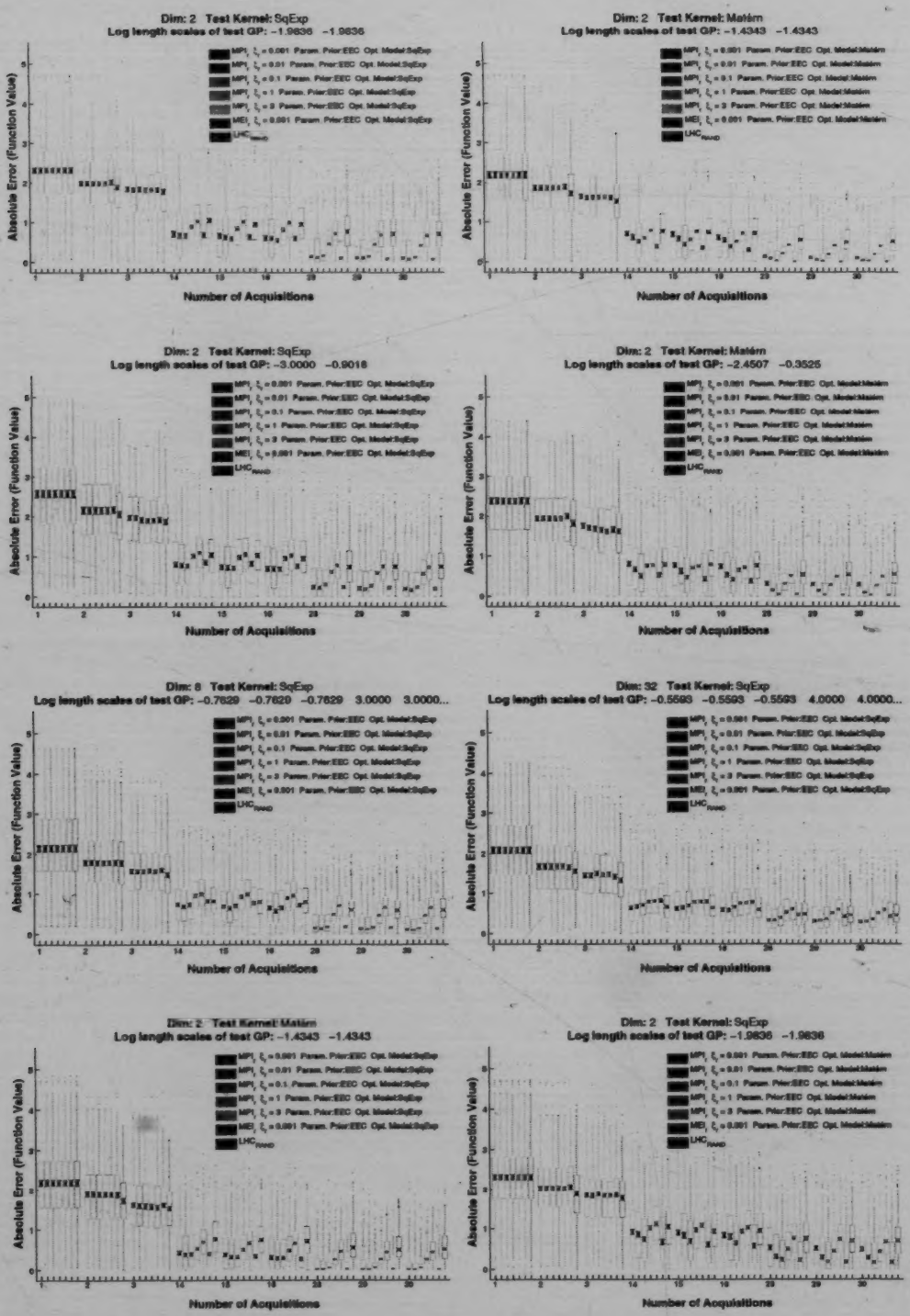

Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −3.0000 −0.9018
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −2.4507 −0.3525
Dim: 8 Test Kernel: SqExp
Log length scales of test GP: −0.7629 −0.7629 −0.7629 3.0000 3.0000...
Dim: 32 Test Kernel: SqExp
Log length scales of test GP: −0.5593 −0.5593 −0.5593 4.0000 4.0000...
Dim: 2 Test Kernel: Matérn
Log length scales of test GP: −1.4343 −1.4343
Dim: 2 Test Kernel: SqExp
Log length scales of test GP: −1.9836 −1.9836
Absolute Error (Function Value)
Number of Acquisitions

Figure B.6: Performance using EEC prior

B.2 With gradient observations

The graphs in this section illustrate the effect of different ξ_r on performance of MEI_r and MPI_r on two test models. The observation models here were built using observed function values and observed gradient values.

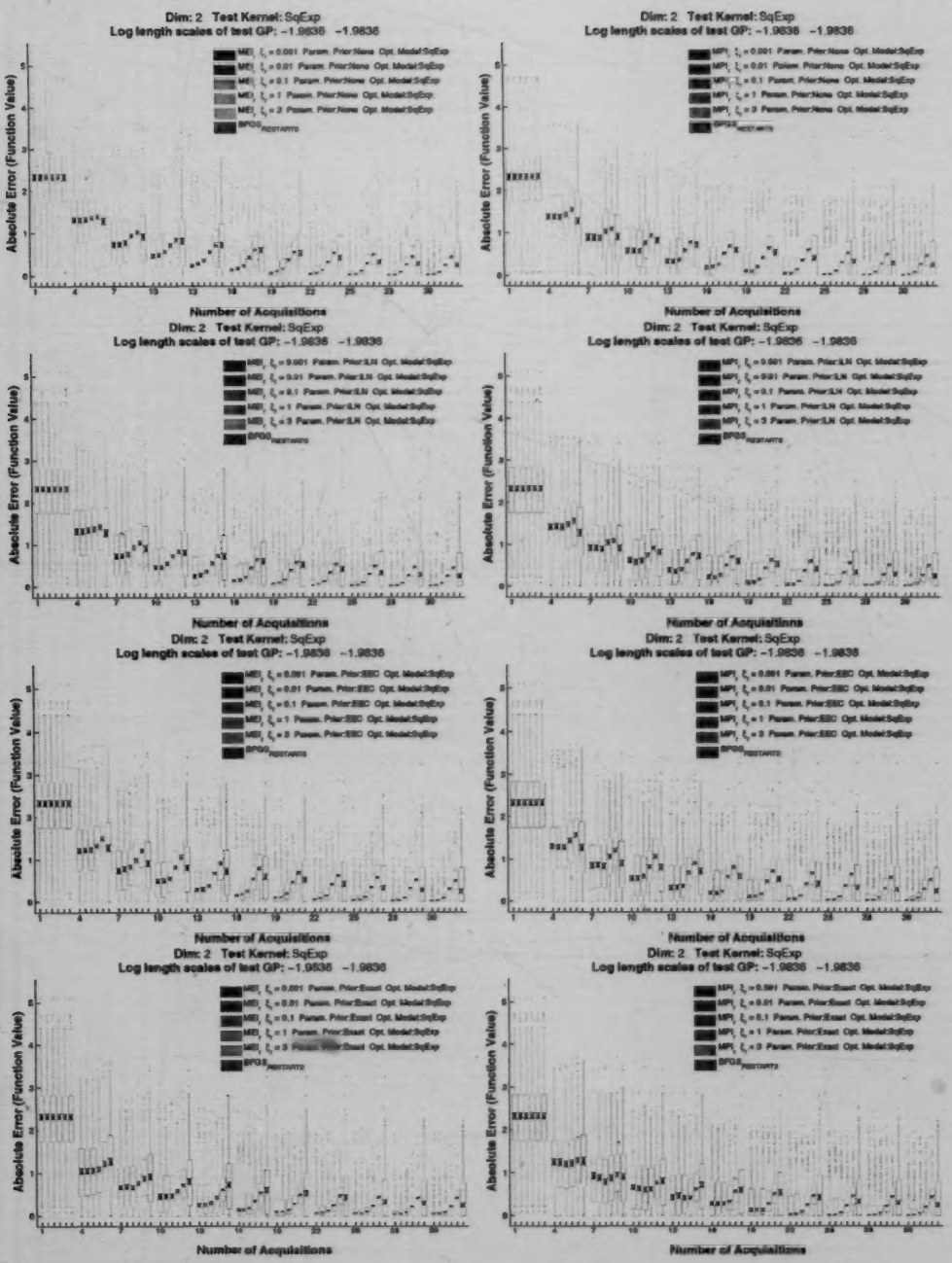


Figure B.7: Performance when using observed function values and gradients. The test kernel is 2D squared exponential with equal length scales. The green boxes illustrate the performance of using the BFGS algorithm with random restarts, but a restricted number of function evaluations.

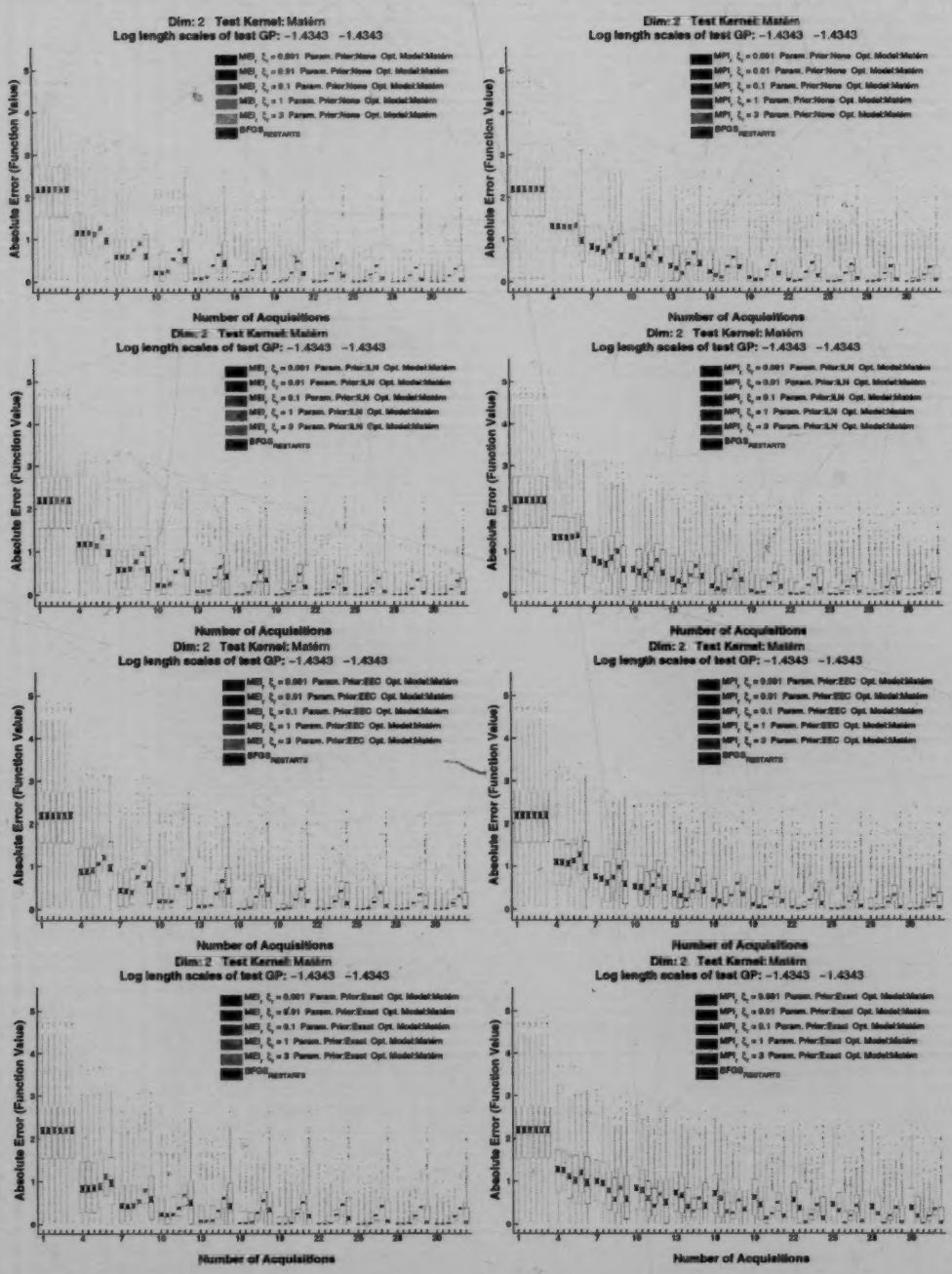


Figure B.8: Performance when using observed function values and gradients. The test kernel is 2D Matérn with equal length scales. The green boxes illustrate the performance of using the BFGS algorithm with random restarts, but a restricted number of function evaluations.

Bibliography

[1] Robert J. Adler. *The Geometry of Random Fields.* Wiley, 1981.

[2] Robert J. Adler and Jonathan E. Taylor. *Random Fields and Geometry.* Springer, 2007.

[3] C. Audet, J.E. Dennis Jr., D.W. Moore, A. Booker, and P.D. Frank. A surrogate-model-based method for constrained optimization. *8th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000.

[4] T. M. D. Bakker. *Design optimization with Kriging models.* Delft University Press, 2000.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, March 2004.

[6] Phillip Boyle. *Gaussian Processes for Regression and Optimisation.* PhD thesis, Victoria University of Wellington, 2006.

[7] R. P. Brent. *Algorithms for Minimization without Derivatives.* Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

[8] Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19(2):263–311, 1993.

[9] Sonia Chernova and Manuela Veloso. An evolutionary approach to gait learning for four-legged robots. In *Intelligent Robots and Systems*, 2004.

[10] L. Csáto and M. Opper. Sparse online gaussian processes. *Neural Computation*, 14(3):641 – 669, 2002.

[11] John F. Elder. Global $\mathbb{R}^d$ optimization when probes are expensive: the GROPE algorithm. In *Proceedings IEEE International Conference on Systems, Man, and Cybernetics*, Chicago, Illinois, 1992.

[12] Daniel E. Finkel. DIRECT - A Global Optimization Algorithm, October 2006. http://www4.ncsu.edu/~ctk/Finkel_Direct/.

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[14] Heiko Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEETPAMI*, 30(2):328 – 341, 2008.

[15] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[16] G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with AIBO. In *IEEE International Conference on Robotics and Automation*, pages 3040–3045, 2000.

[17] G. S. Hornby, M. Fujita, S. Takamura, T. Yamamoto, and O. Hanagata. Autonomous evolution of gaits with the sony quadruped robot. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1297–1304, 1999.

[18] Reiner Horst and Panos M. Pardalos, editors. *Handbook of Global Optimization.* Kluwer Academic Publishers, 1995.

[19] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[20] Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *IJCAI*, pages 1324–1231, 1999.

[21] Benjamin Kedem. *Time Series Analysis by Higher Order Crossings.* IEEE Press, 1993.

[22] M. S. Kim and W. Uther. Automatic gait optimisation for quadruped robots. In *Australasian Conference on Robotics and Automation*, 2003.

[23] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[24] Vladimir Kolmogorov. *Graph Based Algorithms for Scene Reconstruction From Two or More Views.* PhD thesis, Cornell University, January 2004.

[25] H.J. Kushner. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, March 1964.

[26] N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse gaussian process methods: The informative vector machine.

[27] Daniel Lizotte, Tao Wang, Michael Bowling, and Dale Schuurmans. Automatic gait optimization with Gaussian process regression. In *IJCAI*, 2007.

[28] D.J.C. MacKay. Probable networks and plausible predictions - a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems*, 6:469–505, 1995.

[29] Domenico Marinucci. Testing for non-gaussianity on cosmic microwave background radiation: A review. *Statistical Science*, 19(4):294–307, 2004.

[30] Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12-16, 1978.

[31] Daniel Neilson. Personal communication, February 2008.

[32] Isaac Newton. *De analysi per aequationes numero terminorum infinitas.* William Jones, 1669.

[33] C. D. Perttunen. A nonparametric global optimization method using the rank transformation. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 1:888–893, 1989.

[34] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 1992.

[35] Carl Edward Rasmussen. *Advanced Lectures in Machine Learning: ML Summer Schools 2003*, chapter Gaussian Processes in Machine Learning. Springer-Verlag, 2004.

[36] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[37] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(409-435), 1989.

[38] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments.* Springer, 2003.

[39] Michael James Sasena. *Flexibility and Efficiency Enhancements for Constrained Global Design Optimization with Kriging Approximations.* PhD thesis, University of Michigan, 2002.

[40] Daniel Scharstein and Richard Szeliski. http://vision.middlebury.edu/stereo.

[41] Matthias Schonlau. *Computer Experiments and Global Optimization.* PhD thesis, University of Waterloo, 1997.

[42] John Shawe-Taylor and Nello Cristiani. *Kernel methods for pattern analysis.* Cambridge, 2004.

[43] Steven S. Skiena. *The algorithm design manual.* Springer-Verlag New York, Inc., New York, NY, USA, 1998.

[44] B.E. Stuckman. A global search method for optimizing nonlinear systems. *Proceedings of the IEEE Conference on Systems, Man, and Cybernetics*, 1:965–977, 1989.

[45] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *ICML 2005*, Bonn, 2005.

[46] C. Williams. Prediction with Gaussian processes. In *Learning in Graphical Models.* MIT Press, 1999.